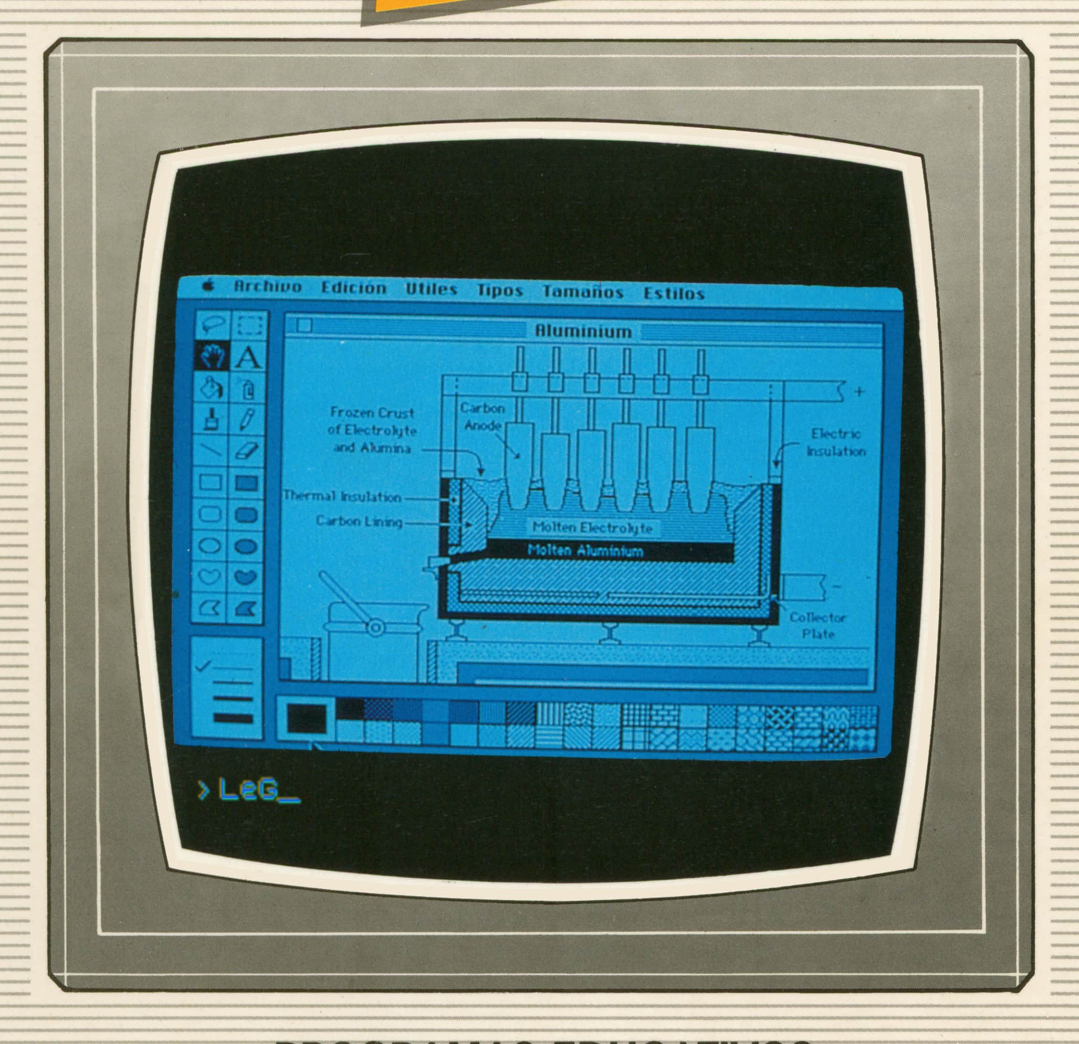
▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 27

PASO A PASO



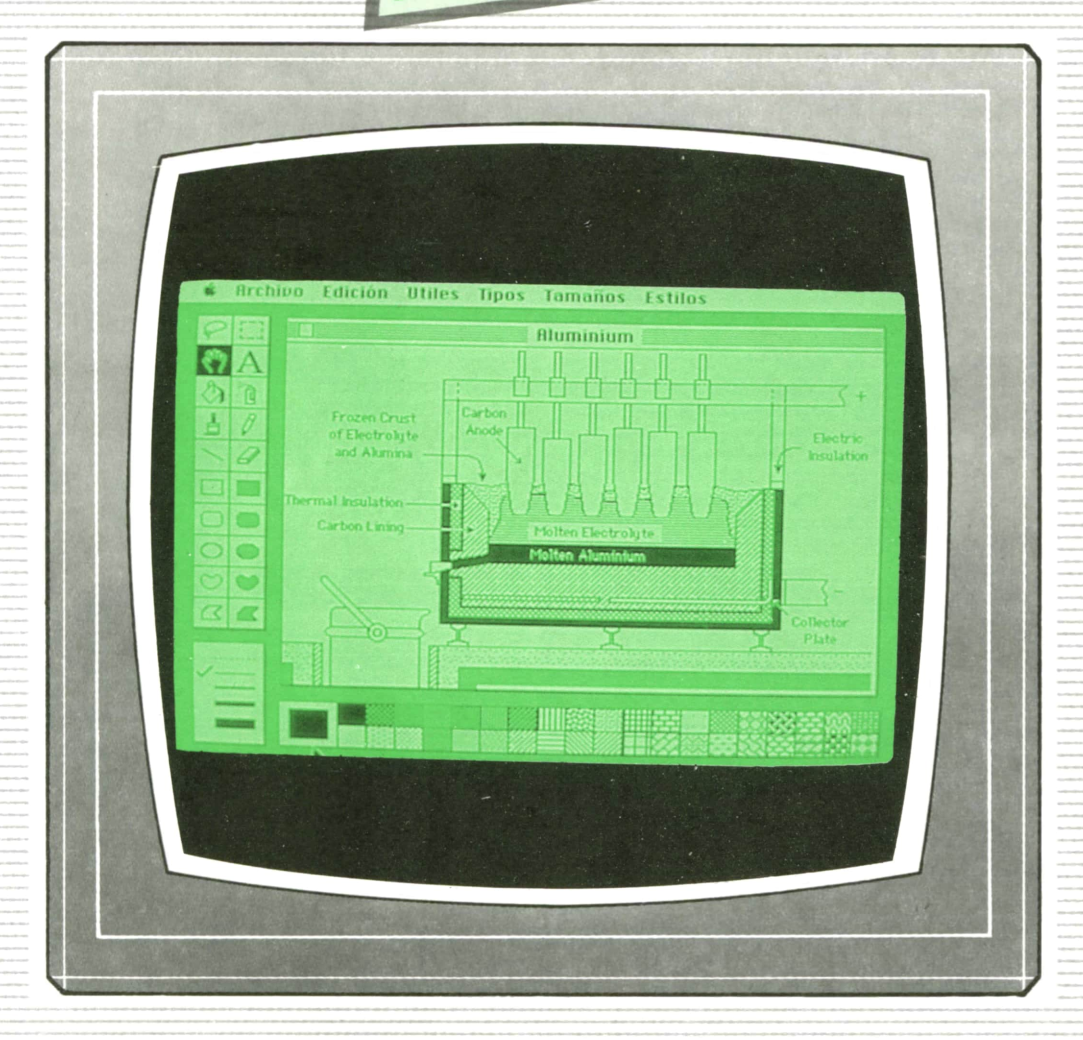
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 27

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas, Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de Aula de Informática Aplicada (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-170-3
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 5.677-1987
Printed in Spain - Impreso en España.

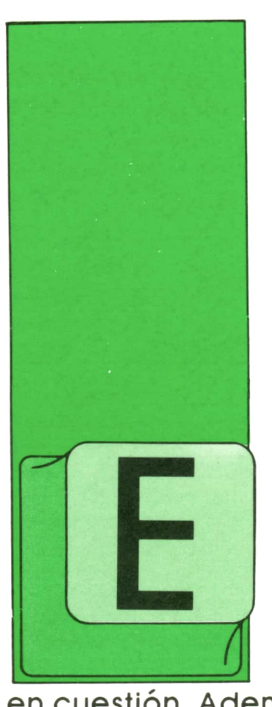
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

Noviembre, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC

LOS DATOS EN LINEAS DE PROGRAMA



En muchas ocasiones los datos que necesita manejar un programa son fijos, por lo que puede resultar muy útil que dichos datos puedan formar parte del programa en cuestión. Además, si el número de datos es alto, las instrucciones LET e INPUT resultan poco efectivas.

El lenguaje BASIC dispone de dos instrucciones que solucionan estos problemas: READ y DATA.

Ambas instrucciones trabajan siempre juntas, son complementarias.

Las instrucciones DATA son simples almacenes de datos, numéricos o alfanuméricos, separados entre sí por comas. Dichos datos deben ser siempre constantes, por tanto, el formato general de DATA se puede expresar:

DATA <lista de datos constantes>

Veamos algunos ejemplos concretos:

```
100 DATA 64,2,32,4,16,8
200 DATA ALEJANDRO,ERNESTO,ESTHER,JAVIER
300 DATA 10,NIEVES,-5,LAURA,28.8
```

Podemos observar que los datos alfanuméricos almacenados en líneas DATA no necesitan ir encerrados entre comillas. Esto sólo será necesario si queremos que una cadena contenga una coma, por ejemplo:

```
100 DATA "SERRANO, 20", PRINCESA, 18"
```

Sin embargo, en el SPECTRUM siempre son necesarias las comillas.

Las instrucciones DATA, también llamadas líneas DATA, tienen las siguientes características.

— No son instrucciones ejecutables. El programa pasa por ellas sin hacer nada.

— En un programa podemos situar tantas líneas DATA como deseemos. Además, podemos ponerlas en cualquier parte del programa sin que esto afecte a la ejecución. Sin embargo, por una cuestión de orden y claridad, conviene que las situemos todas juntas, al principio o al final del programa.

— Las instrucciones DATA de un programa forman un bloque único de datos, por lo que da lo mismo cómo repartamos los datos en las líneas DATA. Por ejemplo, la instrucción:

```
10 DATA 1,3,6,10,15,21,28,36,45
```

es equivalente al conjunto de líneas:

```
10 DATA   1,3,6,10
20 DATA   15,21
30 DATA   28,36,45
```

La instrucción READ sirve para leer los datos almacenados en líneas DATA. Su formato general es el siguiente:

READ <lista de variables>

Cuando el ordenador encuentra una instrucción READ durante la ejecución de un programa busca una línea DATA de la cual leer datos, por eso es indiferente el lugar donde se encuentren las líneas DATA, ya que READ las buscará.

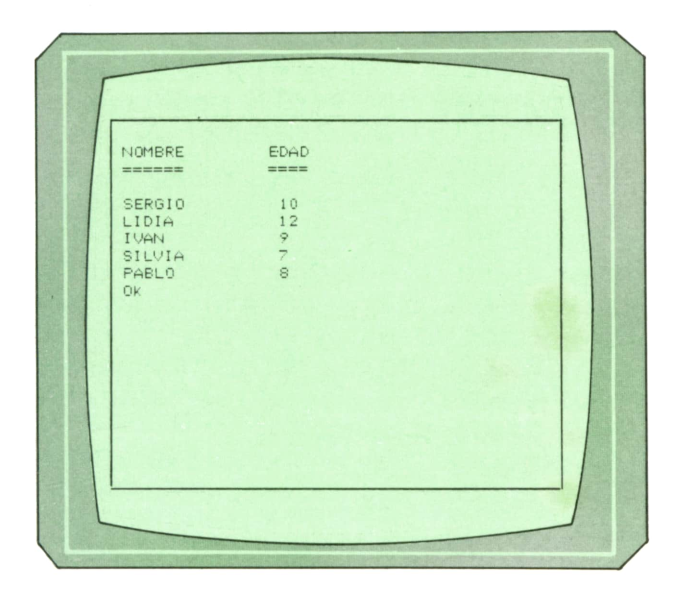
Sin embargo, ¿qué dato leerá READ de todos los almacenados en las líneas DATA? La instrucción READ manda al ordenador que lea a partir del primer dato no leído todavía siguiendo una orden creciente de número de línea de las instrucciones DATA.

Veamos un ejemplo. El programa 1 lee todos los nombres y edades almacenados en las líneas DATA, imprimiéndolos en pantalla.

```
10 REM ***********************
20 REM *  NOMBRES Y EDADES  *
30 REM ***********************
40 CLS
50 PRINT "NOMBRE","EDAD"
60 PRINT "======","===="
70 PRINT
80 FOR I=1 TO 5
90 READ N$,E
100 PRINT N$,E
110 NEXT I
120 DATA SERGIO,10,LIDIA,12
130 DATA IVAN,9,SILVIA
140 DATA 7,PABLO,8
```

El bucle FOR-NEXT se repite cinco veces y en cada vuelta lee e imprime los datos: el primero alfanumérico, que se almacena en N$, y el segundo numérico, almacenado en E. Por tanto, en la primera vuelta del bucle, la instrucción READ leerá los datos SERGIO y 10. En la vuelta siguiente el primer dato no leído todavía es LIDIA, por tanto, leerá LIDIA y 12.

Este proceso se repite cinco veces, por lo que al finalizar la ejecución se habrán leído los diez datos almacenados en las líneas DATA (cinco nombres y cinco edades) y aparecerán en pantalla tal y como muestra la figura 1.





Presentación en pantalla del programa 1.

Hay que hacer notar que el tipo de variables que aparecen en el READ ha de estar en consonancia con el tipo de datos a leer, es decir, si se va a leer un dato numérico, la variable deberá ser numérica, mientras que si el dato es alfanumérico, la variable tendrá que ser alfanumérica. Si no se cumplen estas condiciones obtendremos un mensaje de error. Podemos comprobarlo cambiando el orden de las variables en la línea 90 del programa 1 de la siguiente forma:

90 READ E, N$

Por otra parte, también aparecerá un mensaje de error si intentamos leer con la instrucción READ más datos de los contenidos en las líneas DATA. Sin embargo, no importa que se lean menos datos de los que hay.

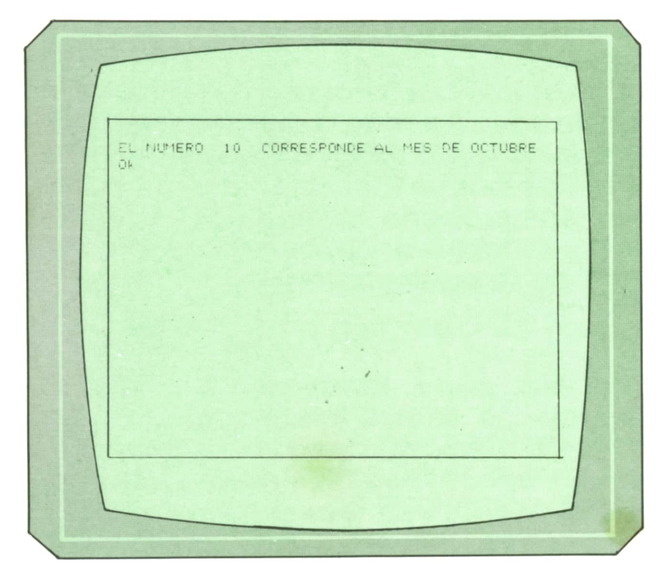
Veamos otro ejemplo. El programa 2 nos permite averiguar el mes correspondiente a un número introducido por teclado.

```
10 REM *************
20 REM *   MESES   *
30 REM *************
40 CLS
50 INPUT "DIME UN NUMERO ENTRE
   1 Y 12";N
60 IF N<1 OR N>12 THEN GOTO 50
70 CLS
80 FOR I=1 TO N
90 READ M$
100 NEXT I
110 PRINT "EL NUMERO ";N;"
    CORRESPONDE AL MES DE ";M$
120 DATA ENERO,FEBRERO,MARZO,
    ABRIL
130 DATA MAYO,JUNIO,JULIO,
    AGOSTO
140 DATA SEPTIEMBRE,OCTUBRE,
    NOVIEMBRE,DICIEMBRE
```

El bucle FOR-NEXT, que se repite N veces, permite que la instrucción READ vaya leyendo uno a uno los nombres de los meses almacenados en líneas DATA.

Al salir del bucle, en la variable M$ se hallará el último mes leído, que se encontrará en la posición N de la lista de datos contenidos en las líneas DATA; por tanto, ese mes será el que corresponda al número N.

En la figura 2 podemos ver un ejemplo de ejecución de este programa.





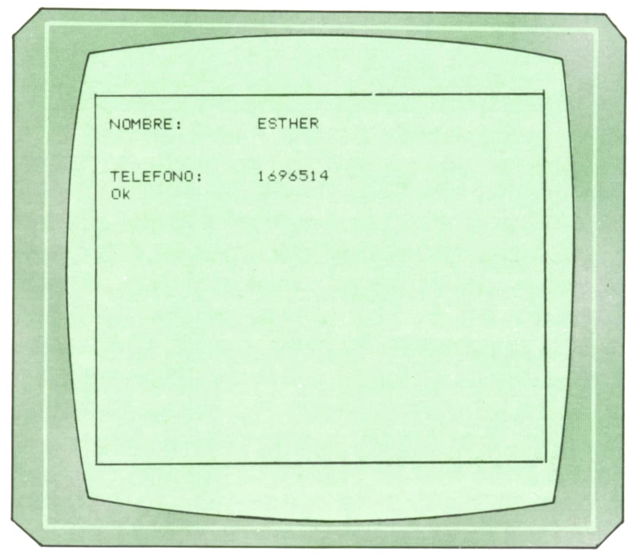
Presentación en pantalla del programa 2.

El programa 3 es otro ejemplo de utilización de READ y DATA. En este caso nos permite localizar el teléfono de la persona que deseemos en una pequeña agenda telefónica.

```
10 REM ***********************
20 REM *  AGENDA TELEFONICA   *
30 REM ***********************
40 CLS
50 INPUT "NOMBRE DE LA PERSONA CUYO
   TELEFONO QUIERES SABER";N$
60 CLS
70 FOR I=1 TO 10
80 READ P$,T
90 IF P$=N$ THEN GOTO 120
100 NEXT I
110 PRINT N$;" NO FIGURA EN LA AGENDA"
   :END
120 PRINT "NOMBRE:",P$
130 PRINT :PRINT
140 PRINT "TELEFONO:",T
150 DATA ALEJANDRO,7910542,ANA,6084504
160 DATA BRUNO,1593517,ELENA,5304574
170 DATA ESTHER,1696514,IGNACIO,7987542
180 DATA JAVIER,9608034,LUIS,4884952
190 DATA MERCEDES,5537614,NIEVES,1257954
```

En cada vuelta del bucle FOR-NEXT la instrucción READ se encarga de leer un nombre y un teléfono de las líneas DATA. La condición de la línea 90 se encarga de comprobar si el nombre leído es igual que el nombre que hemos dicho, en cuyo caso aparecerá en pantalla junto con el teléfono correspondiente (líneas 120-140). En caso de que el bucle dé todas las vueltas sin que se verifique la condición, aparecerá en pantalla el mensaje de la línea 110 advirtiéndonos que el nombre que hemos tecleado no figura en la agenda almacenada en las líneas DATA.

En la figura 3 podemos ver el aspecto de la pantalla tras una posible ejecución de este programa.





Presentación en pantalla del programa 3.



Restore

Hasta ahora hemos visto cómo los datos de las líneas DATA, leídos con READ, se «agotaban», es decir, no se podían volver a leer. Sin embargo, existe una instrucción BASIC que permite la «relectura» de los datos ya leídos: RESTORE.

RESTORE permite que los datos ya leídos en las líneas DATA puedan ser leídos de nuevo desde el principio.

Sin embargo, hay un formato de RESTORE que lo hace todavía más potente:

RESTORE N.° de línea

Este nuevo formato permite seleccionar la línea DATA a partir de la cual deseamos realizar la lectura de datos.

Veamos un ejemplo. El programa 4 imprime en pantalla el presente de indicativo de un verbo regular cualquiera introducido por teclado.

```
10 REM ****************************
20 REM *  PRESENTE DE INDICATIVO  *
30 REM ****************************
40 CLS
50 INPUT "DIME UÑ VERBO REGULAR";V$
60 CLS:PRINT "VERBO:",V$:PRINT
70 LET L=LEN(V$)
80 LET R$=LEFT$(V$,L-2)
90 LET D$=RIGHT$(V$,2)
100 IF D$="AR" THEN RESTORE 170
110 IF D$="ER" THEN RESTORE 190
120 IF D$="IR" THEN RESTORE 210
130 FOR I=1 TO 6
140 READ P$,T$
150 PRINT P$,R$+T$
160 NEXT I
170 DATA YO,O,TU,AS,EL,A
180 DATA NOSOTROS,AMOS,VOSOTROS,AIS,ELLOS,AN
190 DATA YO,O,TU,ES,EL,E
200 DATA NOSOTROS,EMOS,VOSOTROS,EIS,ELLOS,EN
210 DATA YO,O,TU,ES,EL,E
220 DATA NOSOTROS,IMOS,VOSOTROS,IS,ELLOS,EN
```

En el programa utilizamos la fragmentación de cadenas para separar la raíz del verbo (línea 80) de su desinencia (línea 90).

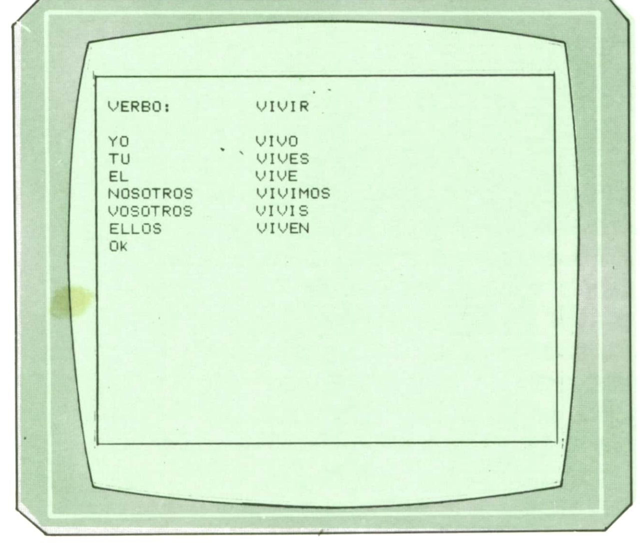
Las condiciones de las líneas 100 a 120 sirven para averiguar de qué conjugación es el verbo, ya que dependiendo de la conjugación las terminaciones para formar el presente serán distintas.

Por tanto, si se verifica la primera condición (línea 100) se podrán leer datos a partir de la línea DATA 170 (terminaciones de la primera conjugación). Si se cumple la segunda condición (línea 110) sólo se podrán leer datos a partir de la línea DATA 190 (terminaciones de la segunda conjugacion). Si se cumple la tercera condición (línea 120), únicamente será posible leer datos a partir de la línea DATA 210 (terminaciones de la tercera conjugación).

El bucle FOR-NEXT (líneas 130-160) se repite seis veces (tantas como número de personas) y en cada vuelta lee de la línea DATA correspondiente la persona y su terminación (línea 140) para, seguidamente, imprimir en pantalla la conjugación correspondiente.

En la figura 4 podemos ver la presentación en pantalla tras una posible ejecución.

Hay que advertir que el COMMODORE no dispone de este formato para RESTORE, por lo que no es posible seleccionar la línea DATA que se desea leer.





Presentación en pantalla del programa 4.

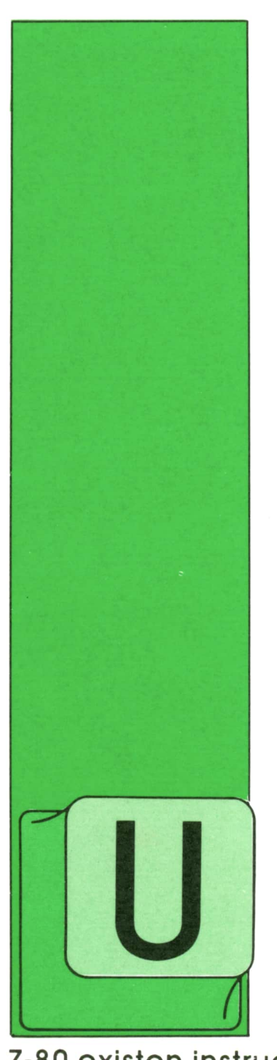
Finalmente, para que este último programa funcione en el SPECTRUM habrá que realizar los siguientes cambios:

```
80 LET R$ = V$ (1 TO L−2)
90 LET D$ = V$ (L−1 TO L)
```

Además, tendremos que poner todos los datos de las líneas DATA entre comillas.

MAQUINA Z-80

SPECTRUM, AMSTRAD, MSX



Instrucciones de entrada/salida

UNA de las principales funciones que tiene que realizar un microprocesador es la de entrada/salida. Existen diversas formas de implementar estas instrucciones. En el Z-80 existen instrucciones especializadas para esta función. Puesto que pueden existir diversos dispositivos que realicen funciones de entrada/salida (pantalla, teclado, casete, impresora, etc.), cada uno estaría identificado por un número de 8 bits, por lo que sólo podrán existir un máximo de 256 dispositivos, lo que es suficiente en la mayor parte de los casos.

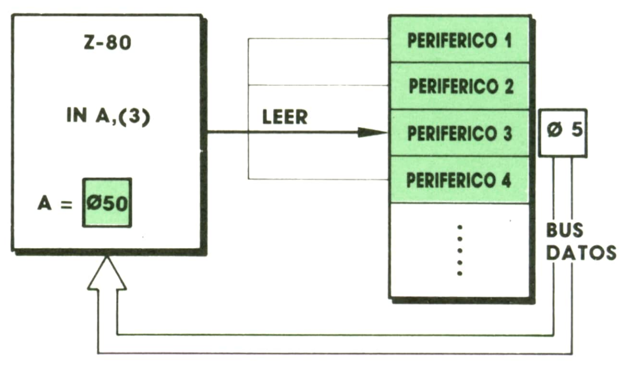
Operaciones de entrada

Estas instrucciones indican al dispositivo que se quiere realizar una operación de entrada, recogiendo el dato que éste proporcione, y almacenándolo en el registro que se indique.

La instrucción más usada es:

IN A, (n)

donde *n* es el número que identifica al dispositivo desde el que se quiere realizar la lectura. El dato leído es almacenado en el acumulador.



EJECUCION IN A,(3)

También se puede seleccionar el dispositivo indirectamente mediante el contenido del registro C:

IN R, (C)

Cargándose el dato en R, donde R representa cualquiera de los registros del Z-80.

Lo más normal no es que tengamos que leer un solo dato, sino que un bloque de ellos; por ello, existen ciertas instrucciones capaces de leer un grupo de datos, al igual que los existentes para mover grupos de datos.

La primera instrucción es:

INI

Esta lee un dato del periférico indicado por el contenido del registro C. Este dato se escribe en la posición de memoria indicada por el contenido del registro HL. Una vez realizada la operación, incrementa el contenido de HL en una unidad y decrementa, también en una unidad, el contenido del registro B.

De esta forma el contenido de HL es una dirección, o puntero, a la zona de memoria donde colocaremos los datos leídos y el registro B sirve como contador del número de datos a leer.

Lo más frecuente es utilizar:

INIR

que ejecuta la instrucción INI repetidamente hasta que el contenido del registro B se hace cero. De esta forma con una sola instrucción podemos leer hasta un bloque de 256 datos (esto es porque el registro B es de 8 bits y, por tanto, sólo puede tomar valores entre 0 y 255).

Existen otras dos instrucciones semejantes a éstas. La primera:

IND

es igual que INI, pero en vez de incrementar el contenido del registro HL lo decrementa, siendo por lo demás idéntica.

También existe la instrucción repetida con:

INDR

Al igual, decrementa HL y B, repitiéndose hasta que B llega a cero.

La diferencia entre las instrucciones que incrementan (INI, INIR) y las que decrementan (IND, INDR) es que las primeras almacenan los datos a partir del contenido inicial del registro doble HL «hacia arriba», uno encima de otro; en los segundos los datos se colocan «hacia abajo», el dato en la posición de memoria anterior a la última.

Grupo de entrada y salida

Código mnemotécnico	Operación simbólica	S	Z		H		P/V	N	C	76	543	210	Hex	N.º de bytes	N.º de ciclos M	N.º de estados T	Comentarios
IN A,(n)	A←(n)	•	•	X	•	X	•	•	•	11 ←n→	011	011	DB	2	3	11	n a A_0-A_7 Acumulador a A_8-A_{15}
IN r,(C)	r←(C) Si r=110, sólo quedan afectados los indicadores	↕	↕	X	↕	X	P	0	•	11 01	101 r	101 000	ED	2	3	12	C a A_0-A_7 B a A_8-A_{15}
INI	(HL)←(C) B←B−1 HL←HL+1	X	↕ (1)	X	X	X	X	1	X	11 10	101 100	101 010	ED A2	2	4	16	C a A_0-A_7 B a A_8-A_{15}
INIR	(HL)←(C) B←B−1 HL←HL+1 Se repite hasta que B=0	X	1	X	X	X	X	1	X	11 10	101 110	101 010	ED B2	2 2	5 (Si B≠0) 4 (Si B=0)	21 16	C a A_0-A_7 B a A_8-A_{15}
IND	(HL)←(C) B←B−1 HL←HL−1	X	↕ (1)	X	X	X	X	1	X	11 10	101 101	101 010	ED AA	2	4	16	C a A_0-A_7 B a A_8-A_{15}
INDR	(HL)←(C) B←B−1 HL←HL−1 Se repite hasta que B=0	X	1	X	X	X	X	1	X	11 10	101 111	101 010	ED BA	2 2	5 (Si B≠0) 4 (Si B=0)	21 16	C a A_0-A_7 B a A_8-A_{15}
OUT (n),A	(n)←A	•	•	X	•	X	•	•	•	11 ←n→	010	011	D3	2	3	11	n a A_0-A_7 Acumulador a A_8-A_{15}
OUT (C),r	(C)←r	•	•	X	•	X	•	•	•	11 01	101 r	101 001	ED	2	3	12	C a A_0-A_7 B a A_8-A_{15}
OUTI	(C)←(HL) B←B−1 HL←HL+1	X	↕ (1)	X	X	X	X	1	X	11 10	101 100	101 011	ED A3	2	4	16	C a A_0-A_7 B a A_8-A_{15}
OTIR	(C)←(HL) B←B−1 HL←HL+1 Se repite hasta que B=0	X	1	X	X	X	X	1	X	11 10	101 110	101 011	ED B3	2 2	5 (Si B≠0) 4 (Si B=0)	21 16	C a A_0-A_7 B a A_8-A_{15}
OUTD	(C)←(HL) B←B−1 HL←HL−1	X	↕ (1)	X	X	X	X	1	X	11 10	101 101	101 011	ED AB	2	4	16	C a A_0-A_7 B a A_8-A_{15}
OTDR	(C)←(HL) B←B−1 HL←HL−1 Se repite hasta que B=0	X	1	X	X	X	X	1	X	11 10	101 111	101 011	ED	2 2	5 (Si B≠0) 4 (Si B=0)	21 16	C a A_0-A_7 B a A_8-A_{15}

NOTAS: (1) Z a 1 si B−1=0; si no, Z a 0.



Operaciones de salida

Este es el caso complementario al anterior. También en este caso se necesita un número para identificar el periférico, normalmente es el mismo número el que identifica a un mismo periférico tanto para entrada como para salida. En este caso también debemos indicar dónde está el dato que queremos escribir en el periférico.

La instrucción principal es:

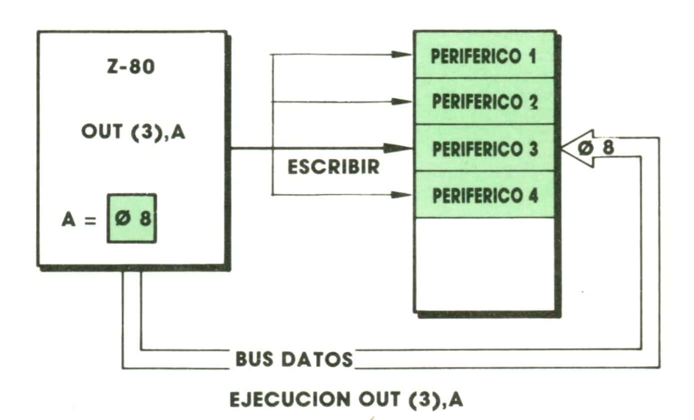
OUT (N),A

donde N es el número que identifica al periférico en el que queremos escribir el dato contenido en el acumulador.

En este tipo de instrucciones también se puede elegir el periférico indirectamente mediante el registro C.

OUT (C),R

donde R es cualquiera de los registros.



EJECUCION OUT (3),A

Además, también están las instrucciones que se ejecutan repetidamente para escribir un bloque de datos.

La instrucción:

OUT1

es idéntica a la INI, con la diferencia de que escribe los datos en vez de leerlos. Análogamente:

OUTIR

ejecuta una instrucción OUTI repetidamente hasta que el contenido de B se anula.

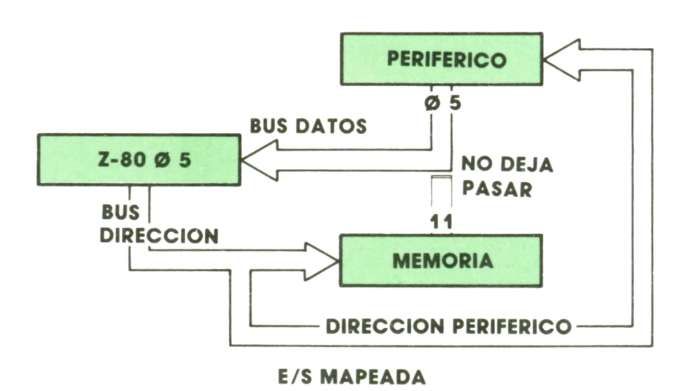
Las dos instrucciones que en vez de incrementar decrementan el contenido HL con OUTD, la que se ejecuta una sola vez, y OUTDR se ejecuta repetidamente.

Existe otra forma de realizar operaciones de entrada/salida conocida como E/S mapeada.

En este caso la dirección del periférico coincide con una dirección de una palabra de la memoria principal.

Al detectar el periférico que se escribe, o lee, de dicha posición coge, o deja el dato de la operación de E/S.

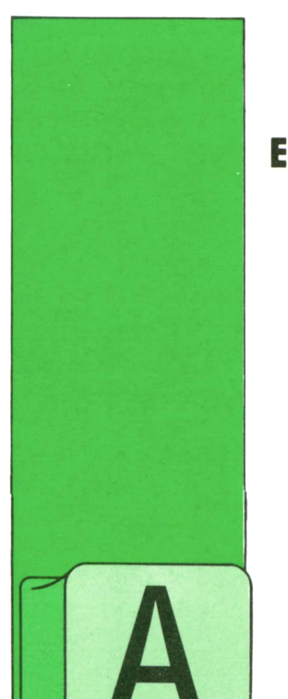
Con este método se amplía en gran manera las posibilidades de realizar las operaciones de entrada/salida.



E/S MAPEADA

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programa: ácidos e hidrácidos

A continuación aparece el segundo programa de química que hemos preparado para esta colección. Si el primero estaba pensado para preguntar al usuario sobre óxidos y anhídridos, este segundo nos permitirá repasar nuestro nivel de ácidos e hidrácidos.

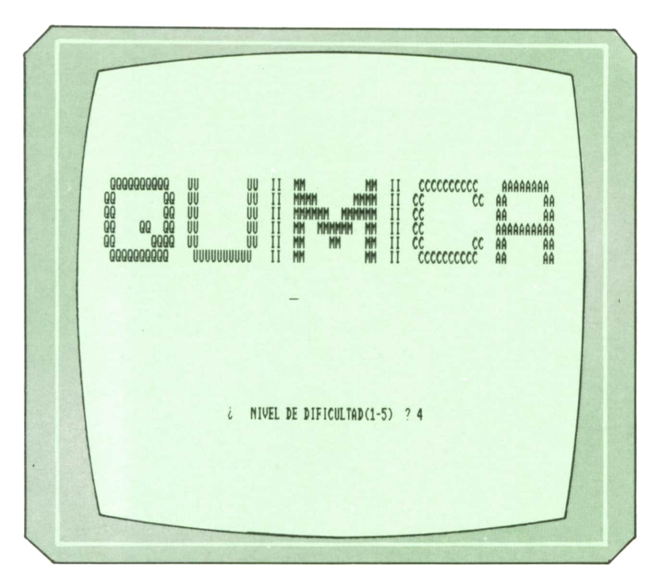




Fig. 1. Presentación y recogida del nivel de dificultad.

El programa nos preguntará el nivel de dificultad del programa y el número de preguntas que queremos que el ordenador nos haga.

A continuación empezarán las preguntas. El ordenador nos preguntará un compuesto y nosotros tendremos que pulsar un número entre 1 y 6 como máximo para responder.

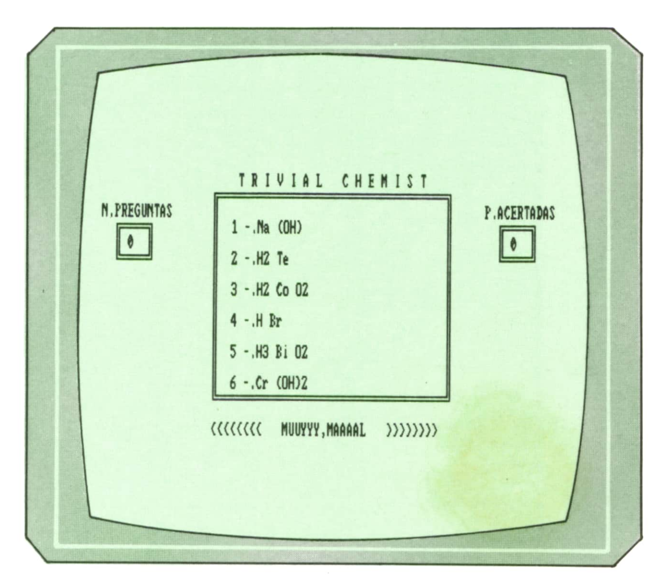




Fig. 2. El programa en plena ejecución.

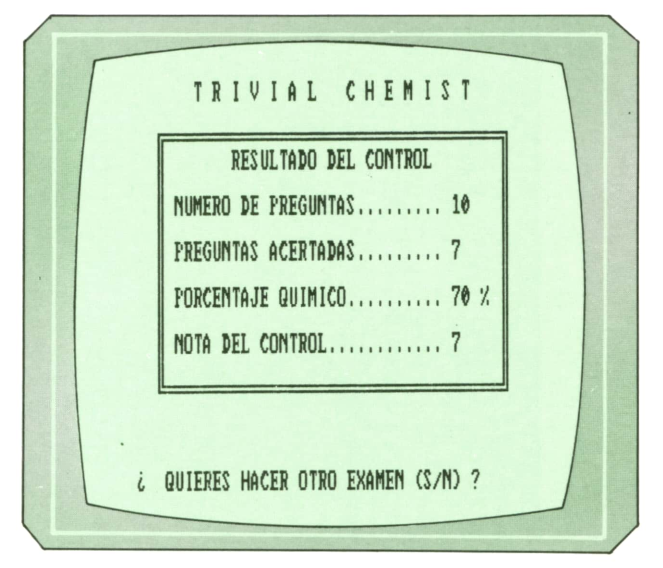




Fig. 3.

Una vez respondidas todas las preguntas, el ordenador nos mostrará por pantalla una breve estadística de nuestra actuación.

```
PROGRAMA: ACIDOS E HIDRACIDOS
=============================

1000 REM ***********************************************************************
1010 REM *************** TRIVIAL CHEMIST POR CARLOS DORAL *******************
1020 REM ***********************************************************************
1030 REM
1040 REM ***********************************************************************
1050 REM ***********     (C) EDICIONES SIGLO CULTURAL  (1987)    **************
1060 REM ***********************************************************************
1070 REM
1080 REM *****************
1090 REM * INICIALIZACION *
1100 REM *****************
1110 REM
1120 SCREEN 0
1130 KEY OFF
1140 COLOR 6
1150 CLS
1160 DIM P$(87)
1170 DIM R$(87)
1180 DIM Q$(8)
1190 DIM H(8)
1200 DIM A(87)
1210 FOR F=1 TO 87
1220    READ P$(F),R$(F)
1230 NEXT F
1240 REM
1250 REM ************************************
1260 REM * PRESENTACION Y RECOGIDA DE NIVEL *
1270 REM ************************************
1280 REM
1290 LOCATE 4,1
1300 PRINT "   QQQQQQQQQQ   UU         UU  II  MM          MM  II   CCCCCCCCCC
 AAAAAAAA      QQ          QQ  UU         UU  II  MMMM      MMMM  II  CC          CC
AA        AA   QQ          QQ  UU         UU  II  MMMMMM  MMMMMM  II  CC
AA        AA"
1310 PRINT "  QQ     QQ  QQ  UU         UU  II  MM   MMMMMM   MM  II  CC
AAAAAAAAAA     QQ       QQQQ  UU         UU  II  MM      MM     MM  II  CC          CC
AA        AA     QQQQQQQQQQ     UUUUUUUUUU    II  MM          MM  II   CCCCCCCCCC
AA        AA"
1320 FOR C=1 TO 50
1330    SOUND 100+INT(RND*800),1
1340 NEXT C
1350 LOCATE 20,26
1360 PRINT "PULSA  /ENTER.  PARA COMENZAR"
1370 IF INKEY$<>CHR$(13) THEN GOTO 1370
1380 LOCATE 20,20
1390 LET ER=0
1400 RANDOMIZE TIMER
1410 INPUT "     (    NIVEL DE DIFICULTAD(1-5)    ";A$
1420 GOSUB 2820
1430 LET ND=VAL(A$)
1440 IF ER=1 OR ND>5 OR ND <1 THEN LOCATE 20,56:PRINT SPACE$(E):GOTO 1380
1450 LET P=0
1460 LET B=0
1470 LET Y=10+(ND*2)
1480 LOCATE 20,21
1490 INPUT "( CUANTAS PREGUNTAS QUIERES (1-87)   ";A$
1500 LET ER=0
1510 GOSUB 2820
1520 LET N=VAL(A$)
1530 IF ER=1 OR N<1 OR N>87 THEN LOCATE 20,50:PRINT SPACE$(E):GOTO 1480
```

```
1540 LOCATE 20,20
1550 COLOR 20
1560 PRINT "  - ATENCION EMPIEZAN LAS PREGUNTAS !     "
1570 COLOR 6
1580 FOR F=1 TO 5
1590    FOR X=1 TO 500
1600    NEXT X
1610    BEEP
1620 NEXT F
1630 REM
1640 REM **********************
1650 REM * PROGRAMA PRINCIPAL *
1660 REM **********************
1670 REM
1680 FOR W=1 TO N
1690    CLS
1700    LET X1=20
1710    LET X2=56
1720    LET Y1=3
1730    LET Y2=8+(ND*2)
1740    GOSUB 3840
1750    LET X1=5
1760    LET X2=10
1770    LET Y1=5
1780    LET Y2=7
1790    GOSUB 3840
1800    LET X1=64
1810    LET X2=69
1820    LET Y1=5
1830    LET Y2=7
1840    GOSUB 3840
1850    LOCATE 4,3
1860    PRINT "N.PREGUNTAS"
1870    LOCATE 4,62
1880    PRINT "P.ACERTADAS"
1890    LOCATE 2,24
1900    PRINT "T R I V I A L   C H E M I S T"
1910    LOCATE 6,6
1920    PRINT P
1930    LOCATE 6,65
1940    PRINT B
1950    LET C=1
1960    LET RN=1+INT(RND*87)
1970    IF A(RN)=1 THEN GOTO 1960
1980    LET A(RN)=1
1990    FOR F=1 TO ND+3
2000       LET H(F)=38+INT(RND*38)
2010    NEXT F
2020    FOR F=1 TO ND+3
2030       FOR C=F+1 TO ND+3
2040          IF H(F)=H(C) OR H(F)=RN THEN GOTO 1990
2050       NEXT C
2060    NEXT F
2070    FOR F=1 TO ND+3
2080       LET Q$(F)=R$(H(F))
2090    NEXT F
2100    LET R=1+INT(RND*(ND+2))
2110    LET Q$(R)=R$(RN)
2120    LET C=1
2130    FOR F=5 TO 8+(ND*2) STEP 2
2140       LOCATE F,22
2150       PRINT C;"-.";Q$(C)
2160       LET C=C+1
2170    NEXT F
2180    LOCATE Y,20
2190    PRINT SPACE$(58)
2200    IF RN<44 THEN M$="ACIDO "+P$(RN)
```

```
2210    IF RN>51 THEN M$="HIDROXIDO "+P$(RN)
2220    IF RN>44 AND RN<52 THEN M$="ACIDO "+P$(RN)+"HIDRICO"
2230    LOCATE Y,20
2240    PRINT "( ";M$;" ?"
2250    LOCATE Y,24+LEN (M$)
2260    LET B$=INKEY$
2270    IF B$<"1" OR B$>CHR$(50+ND) THEN GOTO 2260
2280    IF VAL(B$)=R THEN GOSUB 4320:GOTO 2300
2290    GOSUB 4140
2300    LET P=P+1
2310    LOCATE Y,20
2320    PRINT SPACE$(60)
2330    LOCATE Y,32
2340    PRINT "PULSA UNA TECLA"
2350    IF INKEY$="" THEN GOTO 2350
2360 NEXT W
2370 REM
2380 REM **********************************
2390 REM * PRESENTACION DE LOS RESULTADOS *
2400 REM **********************************
2410 REM
2420 CLS
2430 LET X1=22
2440 LET X2=58
2450 LET Y1=5
2460 LET Y2=16
2470 GOSUB 3840
2480 LOCATE 3,26
2490 PRINT "T R I V I A L   C H E M I S T"
2500 LOCATE 6,30
2510 PRINT "RESULTADO DEL CONTROL"
2520 LOCATE 8,24
2530 PRINT "NUMERO DE PREGUNTAS.........";N
2540 LOCATE 10,24
2550 PRINT "PREGUNTAS ACERTADAS.........";B
2560 LOCATE 12,24
2570 PRINT "PORCENTAJE QUIMICO..........";INT((B*100)/N);"%"
2580 LOCATE 14,24
2590 PRINT "NOTA DEL CONTROL............"
2600 LOCATE 14,52
2610 COLOR 20
2620 PRINT INT((B*10)/N)
2630 COLOR 5
2640 LOCATE 20,20
2650 PRINT "(  QUIERES HACER OTRO EXAMEN (S/N) ?"
2660 LET K$=INKEY$
2670 IF K$="S" OR K$="s" THEN RUN
2680 IF K$="N" OR K$="n" THEN GOTO 2700
2690 GOTO 2660
2700 REM
2710 REM *************
2720 REM * DESPEDIDA *
2730 REM *************
2740 REM
2750 CLS
2760 PRINT "A D I O S ........"
2770 PRINT:PRINT:PRINT
2780 PRINT "ESPERO QUE QUIERAS VOLVER A JUGAR CONMIGO."
2790 PRINT:PRINT:PRINT:PRINT
2800 END
2810 REM
2820 REM ***********************************
2830 REM *** CONTROL DE ERRORES DEL INPUT ***
2840 REM ***********************************
2850 REM
2860 LET E=LEN(A$)
```

```
2870 IF E<1 OR E>2 THEN LET ER=1:RETURN
2880 IF ASC(A$)<49 OR ASC(A$)>56 THEN LET ER=1
2890 RETURN
2900 REM
2910 REM ***********************************************
2920 REM *** DATAS DE LAS PREGUNTAS Y LAS RESPUESTAS ***
2930 REM ***********************************************
2940 REM
2950 DATA HIPOFLUOROSO,H F O
2960 DATA FLUOROSO,H F O2
2970 DATA FLUORICO,H F O3
2980 DATA PERFLUORICO,H F O4
2990 DATA HIPOCLOROSO,H Cl O
3000 DATA CLOROSO,H Cl O2
3010 DATA CLORICO,H Cl O3
3020 DATA PERCLORICO,H Cl O4
3030 DATA HIPOBROMOSO,H Br O
3040 DATA BROMOSO,H Br O2
3050 DATA BROMICO,H Br O3
3060 DATA PERBROMICO,H Br O4
3070 DATA HIPOYODOSO,H I O
3080 DATA YODOSO,H I O2
3090 DATA YODICO,H I O3
3100 DATA PERYODICO,H I O4
3110 DATA HIPOSULFUROSO,H2 S O2
3120 DATA SULFUROSO,H2 S O3
3130 DATA SULFURICO,H2 S O4
3140 DATA HIPOSELENIOSO,H2 Se O2
3150 DATA SELENIOSO,H2 Se O3
3160 DATA SELENICO,H2 Se O4
3170 DATA HIPOTELUROSO,H2 Te O2
3180 DATA TELUROSO,H2 Te O3
3190 DATA TELURICO,H2 Te O4
3200 DATA HIPONITROSO,H3 N O
3210 DATA NITROSO,H3 N O2
3220 DATA NITRICO,H3 N O3
3230 DATA HIPOFOSFOROSO,H3 P O
3240 DATA FOSFOROSO,H3 P O2
3250 DATA FOSFORICO,H3 P O3
3260 DATA HIPOARSENIOSO,H3 As O
3270 DATA ARSENIOSO,H3 As O2
3280 DATA ARSENICO,H3 As O3
3290 DATA HIPOANTIMONIOSO,H3 Sb O
3300 DATA ANTIMONIOSO,H3 Sb O2
3310 DATA ANTIMONICO,H3 Sb O3
3320 DATA HIPOBISMUTOSO,H3 Bi O
3330 DATA BISMUTOSO,H3 Bi O2
3340 DATA BISMUTICO,H3 Bi O3
3350 DATA CARBONOSO,H2 Co O2
3360 DATA CARBONICO,H2 Co O4
3370 DATA SILICIOSO,H2 Si O2
3380 DATA SILICICO,H2 Si O4
3390 DATA CLOR,H Cl
3400 DATA FLUOR,H F
3410 DATA BROM,H Br
3420 DATA YOD,H I
3430 DATA SULF,H2 S
3440 DATA SELEN,H2 Se
3450 DATA TELUR,H2 Te
3460 DATA LITICO,Li (OH)
3470 DATA SODICO,Na (OH)
3480 DATA POTASICO,K (OH)
3490 DATA RUBIDICO,Rb (OH)
3500 DATA CESICO,Cs (OH)
3510 DATA ARGENTIOSO,Ag (OH)
3520 DATA ARGENTICO,Ag (OH)2
```

```
3530 DATA CUPROSO,Cu (OH)
3540 DATA CUPRICO,Cu (OH)2
3550 DATA AUROSO,Au (OH)
3560 DATA AURICO,Au (OH)3
3570 DATA CALCICO,Ca (OH)2
3580 DATA BERILICO,Be (OH)2
3590 DATA ESTRONCICO,Sr (OH)2
3600 DATA BARICO,Ba (OH)2
3610 DATA RADICO,Ra (OH)2
3620 DATA MAGNESICO,Mg (OH)2
3630 DATA ZINCICO,Zn (OH)2
3640 DATA CADMICO,Cd (OH)2
3650 DATA FERROSO,Fe (OH)2
3660 DATA FERRICO,Fe (OH)3
3670 DATA NIQUELOSO,Ni (OH)2
3680 DATA NIQUELICO,Ni (OH)3
3690 DATA CROMOSO,Cr (OH)2
3700 DATA CROMICO,Cr (OH)3
3710 DATA COBALTOSO,Co (OH)2
3720 DATA COBALTICO,Co (OH)3
3730 DATA MANGANOSO,Mn (OH)2
3740 DATA MANGANICO,Mn (OH)3
3750 DATA ALUMINICO,Al (OH)3
3760 DATA BORICO,Bo (OH)3
3770 DATA PLUMBOSO,Pb (OH)2
3780 DATA PLUMBICO,Pb (OH)4
3790 DATA ESTANNOSO,Sn (OH)2
3800 DATA ESTANNICO,Sn (OH)4
3810 DATA PLATINICO,Pt (OH)4
3820 DATA IRIDICO,Ir (OH)4
3830 REM
3840 REM ******************************
3850 REM *** DIBUJO DE LAS VENTANAS ***
3860 REM ******************************
3870 REM
3880 LOCATE Y1,X1
3890 PRINT CHR$(201)
3900 LOCATE Y1,X2
3910 PRINT CHR$(187)
3920 LOCATE Y2,X1
3930 PRINT CHR$(200)
3940 LOCATE Y2,X2
3950 PRINT CHR$(188)
3960 LET F=X1+1
3970 LOCATE Y1,F
3980 PRINT CHR$(205)
3990 LOCATE Y2,F
4000 PRINT CHR$(205)
4010 LET F=F+1
4020 IF F<X2 THEN GOTO 3970
4030 LET F=Y1+1
4040 LOCATE F,X1
4050 PRINT CHR$(186)
4060 LOCATE F,X2
4070 PRINT CHR$(186)
4080 LET F=F+1
4090 IF F<Y2 THEN GOTO 4040
4100 RETURN
4110 REM
4120 REM ***********************
4130 REM *** PREGUNTA MALA ! ***
4140 REM ***********************
4150 REM
4160 LOCATE Y,20
4170 PRINT "<<<<<<<<   MUUYYY,MAAAAL   >>>>>>>>"
4180 FOR F=900 TO 100 STEP -30
4190    SOUND F,1
```

```
4200 NEXT F
4210 LOCATE Y,20
4220 PRINT "  LA RESPUESTA CORRECTA ERA ";R$(RN)
4230 COLOR 20
4240 LOCATE 3+(R*2),27:PRINT R$(RN)
4250 COLOR 5
4260 FOR F=1 TO 3000
4270 NEXT F
4280 RETURN
4290 REM
4300 REM **********************
4310 REM *** PREGUNTA BIEN ! ***
4320 REM **********************
4330 REM
4340 LOCATE Y,20
4350 PRINT "---------- --        MUY,BIEN        !!!!!!!!!"
4360 FOR F=1 TO 100
4370    SOUND 100+(INT(RND*200)),.5
4380 NEXT F
4390 LET B=B+1
4400 FOR F=1 TO 1000
4410 NEXT F
4420 RETURN
```

Agenda telefónica para Spectrum

Normalmente, es muy necesario tener almacenados de alguna manera los teléfonos y las direcciones de mucha gente. Estas direcciones, que no se usan continuamente, aunque no por ello son menos importantes, tienden a perderse y a volvernos locos.

Con este programa podremos almacenar los nombres, dirección y teléfono de todas las personas que queramos para que no se nos pierdan.

El programa nos permite realizar las siguientes funciones:

1. INTRODUCIR FICHAS. Añadir fichas al fichero hasta que le demos orden de parar.
2. MODIFICAR FICHAS. En el caso de que algún conocido cambie de teléfono o de dirección, podemos modificar su ficha perfectamente.
3. VER FICHAS. Podemos ver la ficha o fichas que deseemos en el momento que queramos. El ordenador realiza la búsqueda.

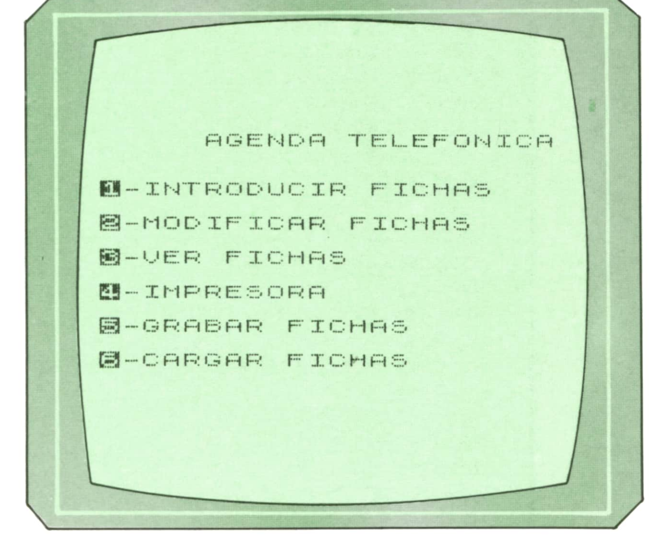



Fig. 4. Menú principal.

4. IMPRESORA. Podemos imprimir la lista de las personas con su dirección y su teléfono por impresora.
5. GRABAR Y CARGAR FICHAS. Podemos grabar el fichero en cinta o en microdrive para más tarde volver a utilizarlo.

6. BORRAR FICHA. En el caso de que no necesitemos cierta dirección, podemos borrarla.

7. ORDENAR AGENDA. Nos permite ordenar toda la agenda por el nombre de la persona.

```
PROGRAMA: AGENDA TELEFONICA

  10 REM ***********************
  11 REM ** AGENDA TELEFONICA **
  13 REM ***********************
  14 REM ********* POR *********
  15 REM ***********************
  16 REM ***** CARLOS DORAL ****
  17 REM ***********************
  20 REM
  21 REM *******************************
  22 REM *(c) Ediciones Siglo Cultural *
  23 REM *(c) 1987.                    *
  24 REM *******************************
  25 REM
  71 INPUT "De cuantas fichas va a constar  el fichero (1-600) ? "; LINE n$
  72 LET e=VAL n$
  74 DIM n$(e,VAL "53")
  75 LET fi=NOT PI
  80 BORDER VAL "1"
  90 PAPER VAL "1"
 100 INK VAL "7"
 101 BRIGHT NOT PI
 110 CLS
 120 PRINT AT NOT PI,VAL "8"; FLASH 1;"AGENDA TELEFONICA"
 130 PRINT AT VAL "3",VAL "3"; INVERSE 1;"1"; INVERSE 0;"-INTRODUCIR FICHAS"
 140 PRINT AT VAL "5",VAL "3"; INVERSE 1;"2"; INVERSE 0;"-MODIFICAR FICHAS"
 150 PRINT AT VAL "7",VAL "3"; INVERSE 1;"3"; INVERSE 0;"-VER FICHAS"
 160 PRINT AT VAL "9",VAL "3"; INVERSE 1;"4"; INVERSE 0;"-IMPRESORA"
 170 PRINT AT VAL "11",VAL "3"; INVERSE 1;"5"; INVERSE 0;"-GRABAR FICHAS"
 180 PRINT AT VAL "13",VAL "3"; INVERSE 1;"6"; INVERSE 0;"-CARGAR FICHAS"
 190 LET k$=INKEY$
 200    IF k$="1" THEN  GO TO VAL "1000"
 210    IF k$="2" AND fi>NOT PI THEN  GO TO VAL "2000"
 220    IF k$="3" AND fi>NOT PI THEN  GO SUB VAL "3000": GO TO VAL "80"
 230    IF k$="4" AND fi>NOT PI THEN  GO SUB VAL "4000": GO TO VAL "80"
 240    IF k$="5" AND fi>NOT PI THEN  GO SUB VAL "5000": GO TO VAL "80"
 250    IF k$="6" THEN  GO TO VAL "6000"
 260 GO TO VAL "190"
 999 REM
1000 REM **********************
1010 REM *  INTRODUCIR FICHAS  *
1020 REM **********************
1025 REM
1030 BORDER NOT PI
1040 PAPER NOT PI
1050 INK VAL "6"
1060 CLS
1070 PRINT AT NOT PI,VAL "8"; FLASH VAL "1";"INTRODUCIR FICHAS"
1080 PRINT AT VAL "2",VAL "20";"FICHAS= ";fi;" "
1085 LET fi=fi+1
1090 PRINT AT VAL "5",NOT PI;"Nombre    : "
1100 PRINT AT VAL "8",NOT PI;"Apellidos : "
1110 PRINT AT VAL "11",NOT PI;"Domicilio : "
1120 PRINT AT VAL "14",NOT PI;"Telefono  : "
1130 INPUT "          "; PAPER VAL "6";"             "; PAPER NOT PI;"              N
ombre= "; LINE m$
1140 IF LEN m$<VAL "1" OR LEN m$>VAL "12" THEN  GO TO VAL "1130"
1141 LET e$="            ": LET m$=m$+e$(LEN m$ TO VAL "11")
1150 PRINT  PAPER VAL "4"; INK NOT PI;AT VAL "5",VAL "12";m$
```

```
1160 INPUT "              "; PAPER VAL "6";"               "; PAPER NOT PI;"          A
pellidos= "; LINE a$
1170 IF LEN a$<VAL "1" OR LEN a$>VAL "12" THEN  GO TO VAL "1160"
1171 LET e$="            ": LET a$=a$+e$(LEN a$ TO VAL "11")
1180 PRINT  PAPER VAL "4"; INK NOT PI;AT VAL "8",VAL "12";a$
1190 INPUT "              "; PAPER VAL "6";"                        "; PAPER NOT PI;" D
omicilio= "; LINE c$
1200 IF LEN c$<VAL "1" OR LEN c$>VAL "20" THEN  GO TO VAL "1190"
1201 LET e$="                    ": LET c$=c$+e$(LEN c$ TO VAL "19")
1210 PRINT  PAPER VAL "4"; INK NOT PI;AT VAL "11",VAL "12";c$
1220 INPUT "             "; PAPER VAL "6";"          "; PAPER NOT PI;"            T
elefono= "; LINE t$
1230 IF LEN t$<VAL "1" OR LEN t$>VAL "9" THEN  GO TO VAL "1220"
1231 LET e$="         ": LET t$=t$+e$(LEN t$ TO VAL "8")
1240 PRINT  PAPER VAL "4"; INK NOT PI;AT VAL "14",VAL "12";t$
1241 LET n$(fi)=m$+a$+c$+t$
1245 PRINT AT VAL "2",VAL "20";"FICHAS= ";fi;" "
1250 PRINT #NOT PI; PAPER VAL "5"; INK NOT PI; BRIGHT VAL "1";"'M' MENU        'S
PACE'-CONTINUAR"
1260 LET k$=INKEY$
1270    IF k$=" " THEN  GO TO VAL "1060"
1280    IF k$="M" OR k$="m" THEN  GO TO VAL "80"
1290 GO TO VAL "1260"
1999 REM
2000 REM ************************
2010 REM *   MODIFICAR FICHAS   *
2020 REM ************************
2025 REM
2030 BORDER NOT PI
2040 PAPER NOT PI
2050 INK VAL "7"
2060 CLS
2070 PRINT AT NOT PI,VAL "10"; FLASH VAL "1";"MODIFICACION"
2080 PRINT AT VAL "5",VAL "5"; INVERSE VAL "1";"1"; INVERSE NOT PI;"-BORRAR FICH
A"
2090 PRINT AT VAL "8",VAL "5"; INVERSE VAL "1";"2"; INVERSE NOT PI;"-MODIFICAR F
ICHAS"
2100 PRINT AT VAL "11",VAL "5"; INVERSE VAL "1";"3"; INVERSE NOT PI;"-ORDENAR AG
ENDA"
2102 PRINT AT VAL "14",VAL "5"; INVERSE VAL "1";"4"; INVERSE NOT PI;"-MENU"
2110 LET k$=INKEY$
2120    IF k$="1" THEN  GO TO VAL "2200"
2130    IF k$="2" THEN  GO TO VAL "2400"
2140    IF k$="3" THEN  GO SUB VAL "2700": GO TO VAL "2000"
2141    IF k$="4" THEN  GO TO VAL "80"
2150 GO TO VAL "2110"
2155 REM
2200 REM **********************
2210 REM *    BORRAR FICHA    *
2220 REM **********************
2225 REM
2230 CLS
2240 PRINT AT NOT PI,VAL "10"; FLASH VAL "1";"BORRAR FICHA"
2250 INPUT "NUMERO DE FICHA A BORRAR (SI NO LO SABES PULSA "; INVERSE 1;"ENTER";
 INVERSE 0;") = "; LINE v$
2260 IF v$="" THEN  GO TO VAL "2370"
2270 FOR f=VAL "1" TO LEN v$
2280    IF CODE v$(f TO f)<VAL "48" OR CODE v$(f TO f)>VAL "57" THEN  GO TO VAL
"2250"
2290 NEXT f
2300 LET n=VAL v$
2310 IF n<VAL "1" OR n>fi THEN  GO TO VAL "2250"
2320 LET n$(n)=""
2330 OUT VAL "254",VAL "7"
2340 CLS
2350 PRINT AT VAL "11",VAL "12"; FLASH VAL "1";"BORRADA"
2360 GO TO VAL "2000"
```

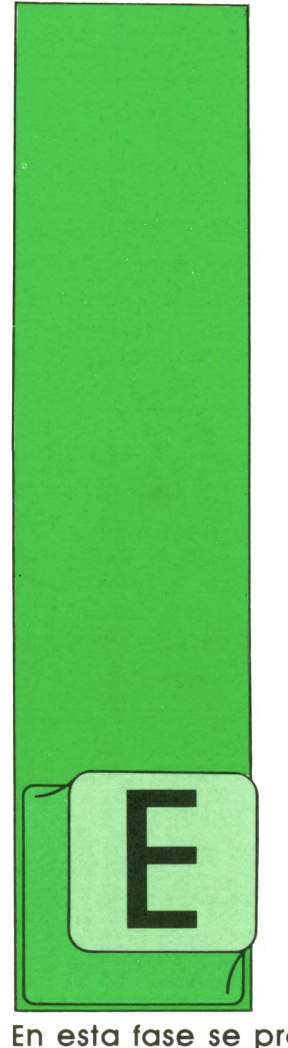
```
2370 INPUT "NOMBRE DE LA PERSONA DE LA FICHAA BORRAR= "; LINE v$
2372 FOR f=VAL "1" TO fi
2373     LET e$=n$(f)
2374     LET e$=e$( TO LEN v$)
2375     IF v$=e$ THEN  GO SUB VAL "9000": OUT VAL "254",VAL "7": CLS : PRINT AT
VAL "11",VAL "12"; FLASH VAL "1";"BORRADA": FOR f=VAL "1" TO VAL "40": NEXT f: G
O TO VAL "2000"
2376 NEXT f
2378 PRINT AT VAL "10",NOT PI;"ESE NOMBRE NO ESTA EN LA AGENDA.PRUEBAS OTRA VEZ
(S/N) ?"
2380 POKE VAL "23658",VAL "8"
2390 LET k$=INKEY$
2392     IF k$="S" THEN  GO TO VAL "2200"
2393     IF k$="N" THEN  GO TO VAL "2000"
2395 GO TO VAL "2380"
2396 REM
2400 REM ***********************
2410 REM *   MODIFICAR FICHAS   *
2420 REM ***********************
2430 CLS
2435 REM
2440 INPUT "NUMERO DE FICHA A MODIFICAR (SI NO LO SABES PULSA "; INVERSE 1;"ENTE
R"; INVERSE 0;")? "; LINE v$
2445 IF v$="" THEN  LET n=VAL "1": GO TO VAL "2570"
2450 FOR f=VAL "1" TO LEN v$
2460     IF CODE v$(f TO f)<VAL "48" OR CODE v$(f TO f)>VAL "57" THEN  GO TO VAL
"2400"
2470 NEXT f
2480 LET n=VAL v$
2490 IF n<VAL "1" OR n>fi THEN  GO TO VAL "2400"
2491 GO TO VAL "2570"
2500 INPUT "NOMBRE DE LA PERSONA PARA LA MO-DIFICACION DE SU FICHA ? "; LINE v$
2510 IF v$="" THEN  GO TO VAL "2500"
2520 FOR f=VAL "1" TO fi
2530     LET j$=n$(f)
2531     LET j$=j$( TO LEN v$)
2535     IF v$=j$ THEN  GO TO VAL "2570"
2540 NEXT f
2550 PRINT AT VAL "10",NOT PI;"ESE NOMBRE NO ESTA EN LA AGENDA.PRUEBAS OTRA VEZ
(S/N) ?"
2560 POKE VAL "23658",VAL "8"
2562 LET k$=INKEY$
2564     IF k$="S" THEN  GO TO VAL "2400"
2566     IF k$="N" THEN  GO TO VAL "2000"
2568 GO TO VAL "2560"
2570 PRINT AT NOT PI,VAL "10"; FLASH VAL "1";"MODIFICACION"
2571 LET e$=n$(n)
2572 LET m$=e$( TO VAL "12")
2573 LET a$=e$(VAL "13" TO VAL "24")
2574 LET c$=e$(VAL "25" TO VAL "44")
2575 LET t$=e$(VAL "45" TO )
2578 PRINT AT VAL "5",NOT PI;"Nombre     : ";m$
2580 PRINT AT VAL "8",NOT PI;"Apellidos : ";a$
2590 PRINT AT VAL "11",NOT PI;"Domicilio : ";c$
2600 PRINT AT VAL "14",NOT PI;"Telefono  : ";t$
2610 INPUT "Nuevo nombre= "; LINE m$
2611 IF LEN m$<VAL "1" OR LEN m$>VAL "12" THEN  GO TO VAL "2610"
2613 LET e$="                    "
2614 LET m$=m$+e$(LEN m$ TO VAL "11")
2620 PRINT  INK NOT PI;AT VAL "5",VAL "12"; PAPER VAL "5"; BRIGHT VAL "1";m$
2630 INPUT "Nuevos apellidos= "; LINE a$
2631 IF LEN a$<VAL "1" OR LEN a$>VAL "12" THEN  GO TO VAL "2630"
2632 LET e$="                    "
2633 LET a$=a$+e$(LEN a$ TO VAL "11")
2640 PRINT  INK NOT PI;AT VAL "8",VAL "12"; PAPER VAL "5"; BRIGHT VAL "1";a$
2650 INPUT "Nuevo domicilio= "; LINE c$
2651 IF LEN c$<VAL "1" OR LEN c$>VAL "20" THEN  GO TO VAL "2650"
```

```
2653 LET e$="                    "
2654 LET c$=c$+e$(LEN c$ TO VAL "19")
2660 PRINT  INK NOT PI;AT VAL "11",VAL "12"; PAPER VAL "5"; BRIGHT VAL "1";c$
2670 INPUT "Nuevo telefono= "; LINE t$
2671 IF LEN t$<VAL "1" OR LEN t$>VAL "9" THEN  GO TO VAL "2670"
2672 LET e$="         "
2673 LET t$=t$+e$(LEN t$ TO VAL "8")
2675 PRINT  INK NOT PI;AT VAL "14",VAL "12"; PAPER VAL "5"; BRIGHT VAL "1";t$
2676 LET n$(n)=m$+a$+c$+t$
2681 PRINT #NOT PI; INVERSE VAL "1";"          ALGUNA MAS(S/N)?         "
2682 POKE VAL "23658",VAL "8"
2683 LET k$=INKEY$
2684    IF k$="S" THEN  GO TO VAL "2400"
2685    IF k$="N" THEN  GO TO VAL "2000"
2686 GO TO VAL "2682"
2689 REM
2700 REM ***************************
2710 REM * ALGORITMO DE ORDENACION *
2720 REM ***************************
2725 REM
2730 BORDER NOT PI
2740 PAPER NOT PI
2750 INK VAL "7"
2760 CLS
2765 PRINT AT NOT PI,VAL "10"; FLASH VAL "1";"ORDENACION"
2770 PRINT AT VAL "15",VAL "8";"PULSA UNA TECLA"
2780 LET k$=INKEY$
2790 IF k$="" THEN  GO TO VAL "2780"
2800 OUT VAL "254",VAL "7"
2810 CLS
2820 PRINT AT VAL "11",VAL "11"; FLASH VAL "1";"ORDENANDO"
2850 FOR c=VAL "1" TO fi-VAL "1"
2860    FOR f=c TO fi
2870       IF n$(c)>n$(f) THEN  GO SUB VAL "2900"
2880    NEXT f
2890 NEXT c
2891 RETURN
2900 LET e$=n$(c)
2910 LET n$(c)=n$(f)
2920 LET n$(f)=e$
2940 RETURN
2999 REM
3000 REM **********************
3010 REM *     VER FICHAS     *
3020 REM **********************
3025 REM
3030 BORDER NOT PI
3040 PAPER NOT PI
3050 INK VAL "7"
3060 CLS
3070 IF fi<=NOT PI THEN  RETURN
3120 INPUT "DESDE QUE NUMERO ("; INVERSE 1;"ENTER"; INVERSE 0;" DESDE ELPRIMERO)
 ? "; LINE v$
3140 FOR f=VAL "1" TO LEN v$
3150    IF CODE v$(f TO f)<VAL "48" OR CODE v$(f TO f)>VAL "57" THEN  GO TO VAL
"3120"
3160 NEXT f
3170 LET n2=fi
3211 IF v$="" THEN  LET n1=VAL "1": GO TO VAL "3231"
3214 IF VAL v$<VAL "1" OR VAL v$>fi THEN  GO TO VAL "3000"
3215 LET n1=VAL v$
3220 LET f=n1
3231    PRINT AT NOT PI,VAL "10"; FLASH VAL "1";"VER FICHAS"
3232    PRINT AT VAL "5",NOT PI;"Nombre     :"
3233    PRINT AT VAL "8",NOT PI;"Apellidos  :"
3234    PRINT AT VAL "11",NOT PI;"Domicilio  :"
3235    PRINT AT VAL "14",NOT PI;"Telefono   :"
```

```
3236    PRINT AT VAL "2",VAL "20";"FICHA= ";f
3237    LET m$=n$(f)
3240    PRINT AT VAL "5",VAL "12";m$( TO VAL "12")
3250    PRINT AT VAL "8",VAL "12";m$(VAL "13" TO VAL "24")
3260    PRINT AT VAL "11",VAL "12";m$(VAL "25" TO VAL "44")
3270    PRINT AT VAL "14",VAL "12";m$(VAL "45" TO )
3280    PRINT #NOT PI;"'A'"; INVERSE 1;"-ADELANTE "; INVERSE 0;"'R'"; INVERSE 1;
"-ATRAS "; INVERSE 0;"'M'"; INVERSE 1;"-MENU"
3290    POKE VAL "23658",VAL "8"
3295    LET k$=INKEY$
3300    IF k$="A" AND f<n2 THEN  LET f=f+1: RANDOMIZE USR 65044: CLS : GO TO VAL
 "3231"
3310    IF k$="R" AND f>VAL "1" THEN  LET f=f-VAL "1": CLS : GO TO VAL "3231"
3320    IF k$="M" OR k$="m" THEN  RETURN
3330 GO TO VAL "3290"
3999 REM
4000 REM **********************
4010 REM *      IMPRESORA      *
4020 REM **********************
4025 REM
4030 BORDER VAL "7"
4040 INK NOT PI
4050 PAPER VAL "7"
4060 CLS
4070 INPUT "DESDE QUE FICHA QUIERES IMPRIMIR ("; INVERSE 1;"ENTER"; INVERSE 0;"
DESDE LA 1) ? "; LINE v$
4071 IF LEN v$>VAL "3" THEN  GO TO VAL "4070"
4080 IF v$="" THEN  LET n1=VAL "1"
4090 INPUT "HASTA QUE FICHA QUIERES IMPRIMIR ("; INVERSE 1;"ENTER"; INVERSE 0;"
HASTA LA ULTIMA) ? "; LINE w$
4091 IF LEN w$>VAL "3" THEN  GO TO VAL "4070"
4100 IF w$="" AND v$="" THEN  LET n2=fi: LET n1=VAL "1": GO TO VAL "4500"
4110 IF v$="" AND w$<>"" THEN  LET n1=VAL "1": GO TO VAL "4132"
4120 FOR f=VAL "1" TO LEN v$
4130    IF CODE v$(f TO f)<VAL "48" OR CODE v$(f TO f)>VAL "57" THEN  GO TO VAL
"4070"
4131 NEXT f
4132 FOR f=VAL "1" TO LEN w$
4140    IF CODE w$(f TO f)<VAL "48" OR CODE w$(f TO f)>VAL "57" THEN  GO TO VAL
"4090"
4150 NEXT f
4160 LET n1=VAL v$
4170 LET n2=VAL w$
4180 IF n1<VAL "1" OR n2>fi THEN  GO TO VAL "4000"
4500 CLS
4502 LPRINT "            AGENDA    "
4510 FOR f=n1 TO n2
4511    LPRINT '
4512    LPRINT "  Ficha num. : ";f
4520    LPRINT
4525    LET m$=n$(f)
4530    LPRINT "Nombre     : ";m$( TO VAL "12")
4535    LET a$=n$(f)
4550    LPRINT "Apellidos : ";a$(VAL "13" TO VAL "24")
4555    LET c$=n$(f)
4570    LPRINT "Domicilio : ";c$(VAL "25" TO VAL "44")
4575    LET t$=n$(f)
4590    LPRINT "Telefono  : ";t$(VAL "45" TO )
4600    LPRINT
4610    LPRINT "     ----------------------"
4620    LPRINT
4630 NEXT f
4640 LPRINT
4650 LPRINT "     **** FIN DE AGENDA ****"
4660 RETURN
4999 REM
5000 REM ***********************
```

```
5010 REM *    GRABAR AGENDA    *
5020 REM ***********************
5025 REM
5030 BORDER NOT PI
5040 PAPER NOT PI
5050 INK VAL "7"
5060 CLS
5070 PRINT AT NOT PI,VAL "11"; FLASH VAL "1";"GRABACION"
5080 PRINT AT VAL "10",VAL "5";"Nombre de fichero:"
5090 INPUT "Nombre= "; LINE e$
5100 IF LEN e$<VAL "1" OR LEN e$>VAL "10" THEN  GO TO VAL "5090"
5110 SAVE e$ DATA n$()
5120 PRINT #NOT PI; PAPER VAL "6"; INK NOT PI;"    QUIERES VERIFICAR (S/N) ?   "
5130 LET k$=INKEY$
5140 IF k$="" THEN  GO TO VAL "5130"
5150 IF k$="S" OR k$="s" THEN  VERIFY "": RETURN
5160 RETURN
5999 REM
6000 REM ***********************
6010 REM *    CARGAR AGENDA    *
6020 REM ***********************
6025 REM
6030 BORDER NOT PI
6040 PAPER NOT PI
6050 INK VAL "7"
6060 CLS
6070 PRINT AT NOT PI,VAL "9"; FLASH VAL "1";"CARGA DE DATOS"
6080 INPUT "Nombre= "; LINE e$
6090 IF LEN e$>VAL "10" THEN  GO TO VAL "6080"
6100 PRINT AT VAL "10",VAL "12"; FLASH VAL "1"; BRIGHT VAL "1";"CARGANDO"
6110 LOAD e$ DATA n$()
6120 RETURN
6998 REM
6999 REM **********************
9000 REM *    BORRAR FICHA    *
9001 REM **********************
9002 REM
9010 IF f=fi THEN  LET n$(f)="": LET fi=fi-VAL "1": RETURN
9020 FOR e=f TO fi-VAL "1"
9030    LET n$(e)=n$(e+VAL "1")
9070 NEXT e
9080 LET fi=fi-VAL "1"
9090 RETURN
```

TECNICAS DE ANALISIS



Estudio detallado de un sistema

Es la fase final de todo el proceso de análisis y diseño. Es la etapa de culminación de la tarea de organización y estructuración de las informaciones y los procedimientos. En esta fase se producen los cuadernos de cargas necesarios para la realización del software correspondiente y para la puesta en marcha de los cambios organizativos que subyacen en las futuras aplicaciones a implementar.

A pesar de ello, es importante controlar que las especificaciones descritas en estos cuadernos de cargas no rebasen el marco en que se han definido las aplicaciones correspondientes, según el plan-masa aprobado. Debe, por tanto, continuar la tarea del comité director del plan, para supervisar estos aspectos del desarrollo de los trabajos.

Se suelen establecer cinco fases sucesivas en el desarrollo del estudio detallado de una aplicación:

a) *Fase de exploración*

En ella se deben reseñar y estructurar todos los casos que haya que tener en cuenta desde el punto de vista de la organización y determinar las dificultades posibles que puedan surgir al aplicar las soluciones previstas en el plan-masa.

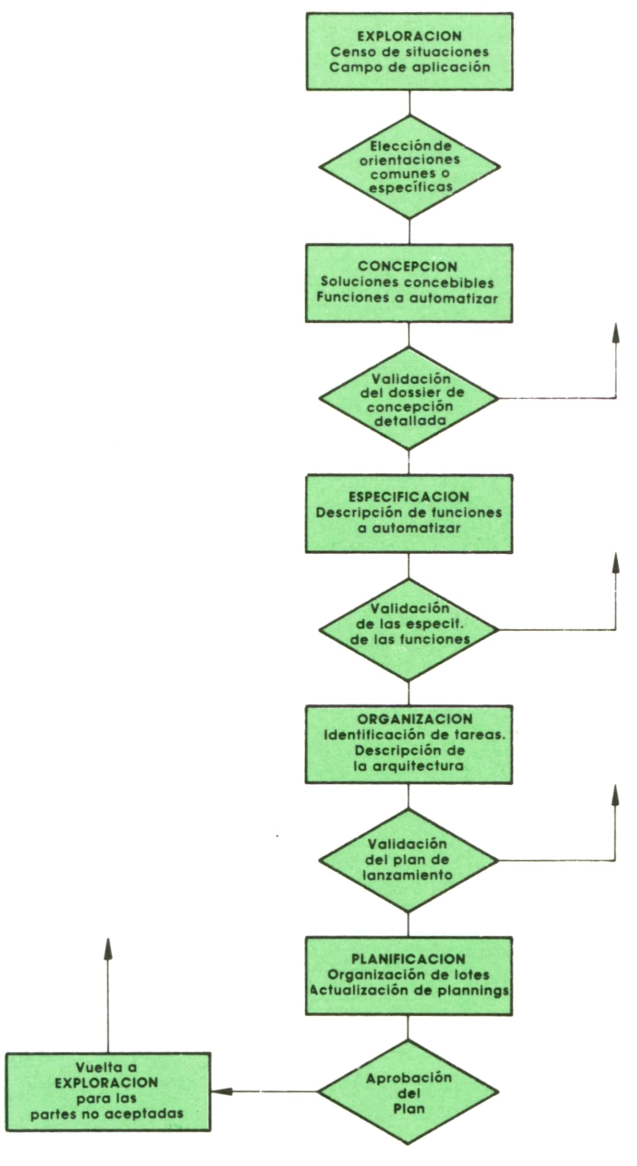
En esta fase se debe investigar la aplicabilidad de los principios de soluciones comunes. Al concluir la fase debe estar fijado el campo de aplicación del estudio y elegidas las orientaciones comunes o específicas para las soluciones que se van a concebir.

b) *Fase de concepción*

En esta fase se definen las diferentes soluciones funcionales concebidas, derivadas de las soluciones-tipo establecidas para resolver todas las situaciones especificadas: una solución puede diferir de otra, quizá, sólo en la modificación de un aspecto organizativo o en la utilización de un medio material diferente. Se debe establecer un censo de entradas, salidas, pantallas, tareas, datos..., que deberán tenerse en cuenta en cada aplicación informática. Hay que detallar las funciones que se han de automatizar.

c) *Fase de especificación*

En esta tercera fase se describen, ante todo, los elementos censados en la etapa precedente (entradas, salidas, pantallas, reglas de control, cálculos, normas de puesta al día, etc.). Usualmente, esta descripción a la que aludimos, se suele hacer bajo la forma de maquetas. Hay que diseñar, además, los puntos básicos de la solución técnica a aplicar). Esta etapa es sumamente importante en el conjunto del proceso, por lo que ha de ponerse atención para que sean validados cuidadosamente todos los elementos descritos, antes de pasar a la realización.



d) *Fase de organización detallada*

Esta es una etapa de estudio técnico de la arquitectura del software que se va a desarrollar y de las bases de datos que se van a implementar.

Esto ha de ser considerado tanto desde el punto de vista del futuro sistema, como desde el del paso de la situación actual a la futura que se desea implementar (aspectos ya considerados, desde el punto de vista funcional, en etapas anteriores). La validación de las soluciones diseñadas en esta fase es eminentemente técnica y debe ser realizada, fundamentalmente, por los servicios de métodos, explotación y sistemas.

e) *Fase de planificación*

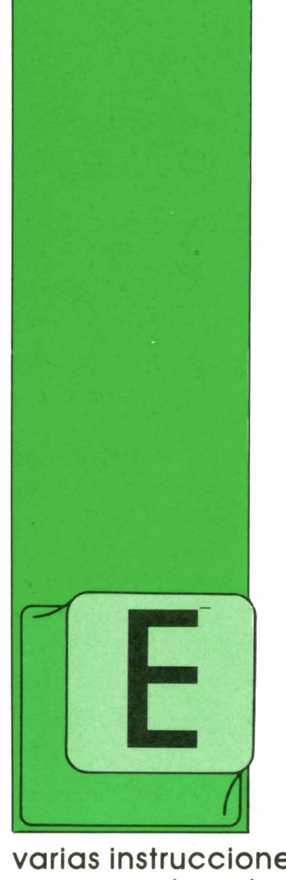
En esta etapa hay que descomponer el trabajo pendiente de desarrollo en lotes (de realización, de puesta en marcha, de asistencia, etc.). A continuación hay que valorar estos lotes y, por último, prever los medios personales y materiales que se van a dedicar a cada tarea.

El resultado de este trabajo es un planning detallado, junto con un presupuesto (detallado también) y actualizado de los costes previsibles.

Como conclusión de todo este proceso, el comité director debe evaluar la tarea desarrollada y aprobar parcial o totalmente los desarrollos propuestos.

Quedan las dos últimas fases del desarrollo de todo proyecto; la realización concreta de las aplicaciones y la puesta en marcha de los procesos mecanizados; son tareas, ambas, sumamente importantes pero que no forman parte, propiamente, del análisis de las soluciones informáticas que hemos comentado.

TECNICAS DE PROGRAMACION



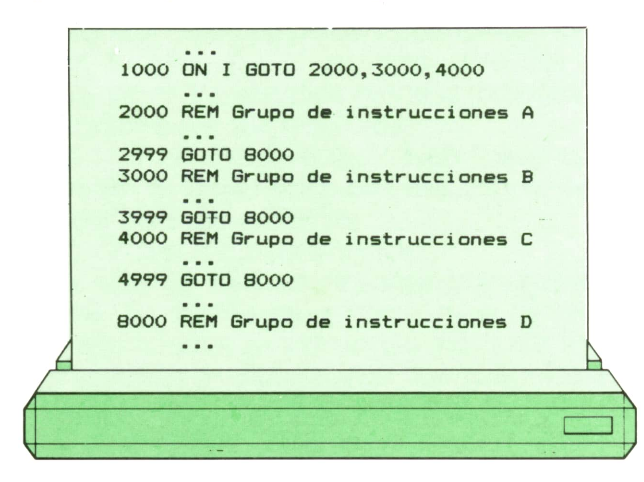


Programación modular (continuación)

EXISTE en Basic una forma especial de la instrucción de llamada a una subrutina o módulo (GOSUB) que permite simplificar cierto tipo de situaciones, reduciendo varias instrucciones a una sola. Supongamos, por ejemplo, que nos encontramos, en nuestro programa, en una instrucción donde tenemos que hacer lo siguiente:

«Si la variable I vale 1, ejecutar el conjunto de instrucciones A. Si I vale 2, ejecutar el conjunto de instrucciones B. Si vale 3, ejecutar el conjunto C. Supongamos, además, que después de ejecutar los grupos de instrucciones indicados, todos los caminos deben reunirse en la instrucción 8000 para continuar la ejecución con el grupo de instrucciones D.»

Hemos visto en capítulos anteriores que esto puede conseguirse así:



```
     ...
1000 ON I GOTO 2000,3000,4000
     ...
2000 REM Grupo de instrucciones A
     ...
2999 GOTO 8000
3000 REM Grupo de instrucciones B
     ...
3999 GOTO 8000
4000 REM Grupo de instrucciones C
     ...
4999 GOTO 8000
     ...
8000 REM Grupo de instrucciones D
     ...
```

El organigrama del programa anterior es el siguiente:

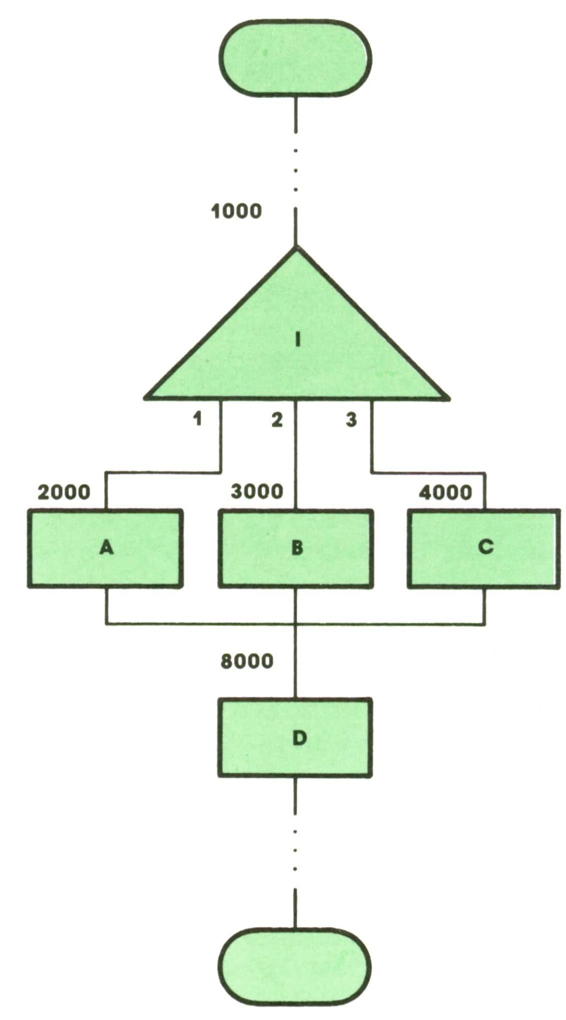
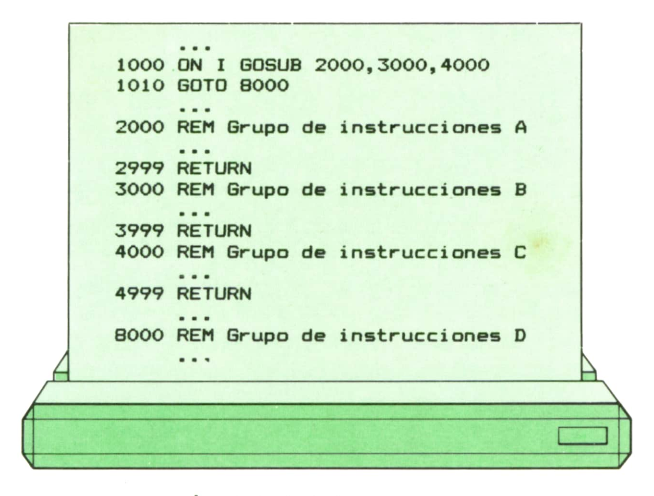




Fig. 1.

Existe una manera más elegante de conseguir el mismo resultado con programación modular. En esta otra forma, el programa anterior quedaría así:



```
     ...
1000 ON I GOSUB 2000,3000,4000
1010 GOTO 8000
     ...
2000 REM Grupo de instrucciones A
     ...
2999 RETURN
3000 REM Grupo de instrucciones B
     ...
3999 RETURN
4000 REM Grupo de instrucciones C
     ...
4999 RETURN
     ...
8000 REM Grupo de instrucciones D
     ...
```

cuyo organigrama es:

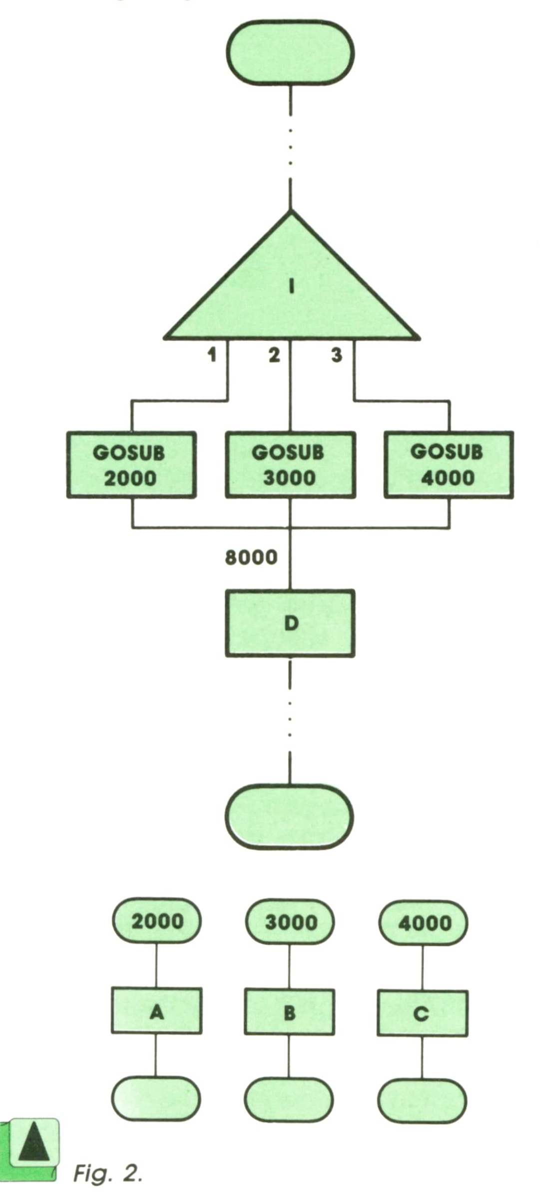
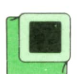


Fig. 2.

Obsérvese que, si colocáramos las instrucciones del grupo D justo detrás de la instrucción 1000 (pasándolas a la etiqueta 1010), podríamos prescindir de la instrucción GOTO, que en el ejemplo aparece con la etiqueta 1010.



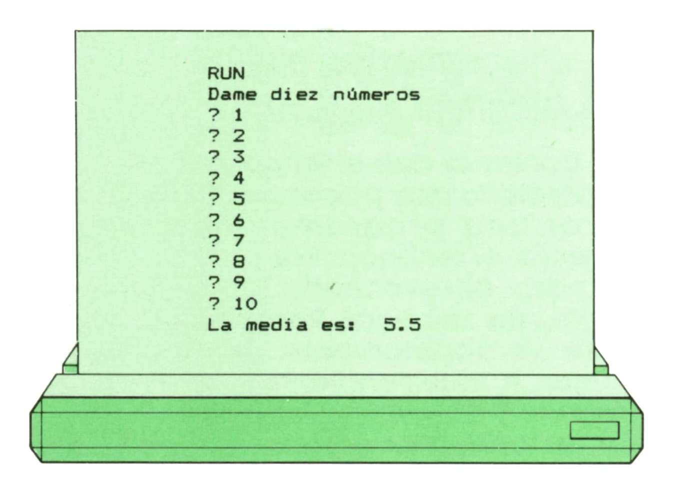
Programación modular en el lenguaje PASCAL

Al contrario que el lenguaje Basic, que proporciona muy pocas posibilidades de hacer una programación modular correcta, el lenguaje Pascal está diseñado para aprovecharla al máximo. En efecto, los módulos Pascal pueden hacerse verdaderamente cerrados en sí mismos e independientes del resto del programa donde se utilizan. Por esta razón, es posible separarlos de dicho programa y utilizarlos directamente en otros sin tomar tantas precauciones como exige la programación en Basic para hacer lo propio. Además, la independencia de los módulos Pascal permite que varios programadores puedan colaborar en un mismo proyecto sin tener que ponerse de acuerdo en los nombres de las variables a utilizar, lo que facilita enormemente la eficacia de su trabajo.

Veamos un ejemplo comparativo, que nos demuestre cuáles son las diferencias fundamentales entre la programación modular Basic y la programación modular Pascal. Supongamos que deseamos construir un módulo que nos calcule la media de N números enteros. En el programa principal leeremos estos números del teclado y los colocaremos en una variable que pueda contener una serie de datos. Luego llamaremos a la subrutina que calcula la media de estos datos y la imprimiremos. El programa podría quedar así:

```
10 REM Este programa lee 10 números
   y calcula su media
20 PRINT "Dame diez números"
30 LET N=10
40 DIM X(N)
50 FOR I=1 TO N: INPUT X(I): NEXT I
60 GOSUB 1000: REM Cálculo de la media
70 PRINT "La media es: ";M
80 END
1000 REM Subrutina que calcula la media
1010 LET M=0
1020 REM Acumulamos los números en M
1030 FOR I=1 TO N: LET M=M+X(I): NEXT I
1040 REM Calculamos su media
1050 LET M=M/N
1060 RETURN
```

Este programa funciona. Veamos, en efecto, cuál es el resultado de su ejecución:



Sin embargo, supongamos que queremos aprovechar la subrutina para incorporarla en otro programa principal diferente donde también necesitemos calcular la media. Nos encontraremos las siguientes dificultades:

1. La subrutina supone que debe calcular la media de los valores contenidos en la variable dimensionada X. Además, el valor de la variable N es igual al número de elementos de X de los que deseamos hacer la media. En tercer lugar, la subrutina pone el resultado (la media) en la variable M. Por último, la variable I se utiliza durante el cálculo. ¿Qué pasaría si nuestro nuevo programa, con el que queremos integrar la subrutina, utilizara estas variables para otra cosa? Nos veríamos obligados a cambiar los nombres de cuatro variables, ya sea en el programa, ya en la subrutina. Esto puede no ser tan sencillo como parece. Si nos olvidamos de cambiar el nombre de una sola de estas variables en un solo lugar, nuestro programa tendrá errores que pueden resultar bastante difíciles de localizar.

2. ¿Qué pasaría si nuestro nuevo programa principal utiliza las etiquetas 1000 a 1060 (las de la subrutina) para otra cosa? Nos veríamos obligados a cambiar las etiquetas, ya sea del programa, ya de la subrutina. Esto también puede traer complicacines y dar lugar a errores molestos y que cuesta mucho eliminar.

Es verdad que, en nuestro ejemplo, la subrutina es lo suficientemente pequeña como para que no sea difícil cambiar las etiquetas o los nombres de las variables. Pero estos mismos problemas los vamos a encontrar cuando intentemos llevar a otro programa una subrutina muy grande, para evitar tener que programarla otra vez. Y es precisamente aquí, cuando más útil nos puede ser la programación modular, donde Basic nos pone dificultades, a veces casi insolubles.

Veamos ahora el mismo ejemplo en Pascal:

```
Program EJEMPLO;
TYPE serie = array [1..10] of real;
Function MEDIA (N:integer; X:serie):
     real;
     {Subrutina que calcula la media
      de N números}
   VAR i:integer; total:real;
   BEGIN
     total:=0;
     for i:=1 to N do total:=total+X[i];
     MEDIA:=total/N
   END;
VAR I:integer; datos:serie;
BEGIN
    {Este programa lee 10 números
     y calcula su media}
    writeln ('Escriba 10 números');
    for i:=1 to 10 do readln(datos[I]);
    writeln ('La media es: ',MEDIA(10,
    datos))
END.
```

Observaremos las siguientes peculiaridades:

1. La instrucción TYPE define un nuevo tipo de datos: la serie de diez datos reales, que nos será útil para definir variables de esta clase en el resto del programa.

2. El programa consta de dos partes claramente determinadas. La primera está encabezada por la palabra reservada Function (función), que da comienzo a la definición de un módulo. Este módulo, en lugar de tener una etiqueta numérica, tiene un nombre (MEDIA), lo que lo hace más mnemotécnico.

3. A continuación del nombre del módulo figura una lista de variables. El primero (N) es una variable entera. El segundo (X) es una serie, es decir, un conjunto de diez datos (pues así hemos definido el tipo serie). Estas variables, que van entre paréntesis después del nombre de la función, se llaman parámetros o argumentos de la función, y constituyen el eslabón entre el módulo y el programa principal.

4. A continuación de la lista de argumentos tenemos dos puntos y la palabra real. Esto nos define el tipo de resultado del módulo o función, que dentro de ésta se reconocerá con el mismo nombre que el asignado a la función (MEDIA).

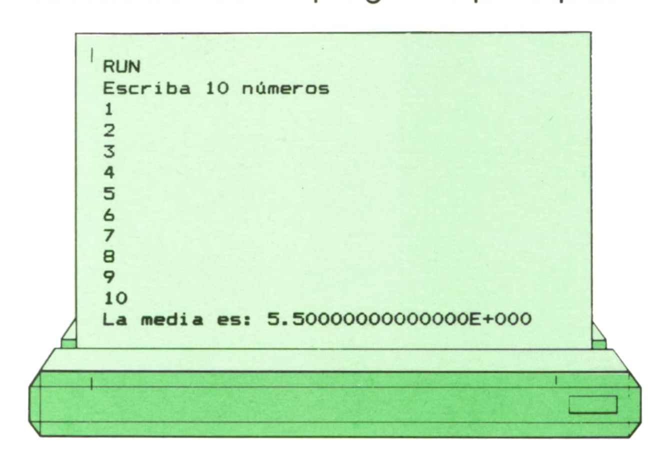
5. A continuación, tenemos una zona declarativa (como en cualquier programa ordinario) que define las variables «locales» a la función. Tanto estas variables como los argumentos o parámetros no entran en conflicto con otras variables del mismo nombre que pueda haber en el programa principal, por lo que se llaman «variables locales» al módulo o función. El único nombre común a ambas zonas (programa principal y módulo) es el nombre del propio módulo (MEDIA, en nuestro caso).

6. Luego viene la parte ejecutable del módulo o función, encerrada entre las palabras reservadas BEGIN y END, como es usual en PASCAL.

7. Después de END comienza la segunda parte de nuestro programa: el programa principal, que también tiene su parte declarativa y su parte ejecutable, como cualquier programa ordinario.

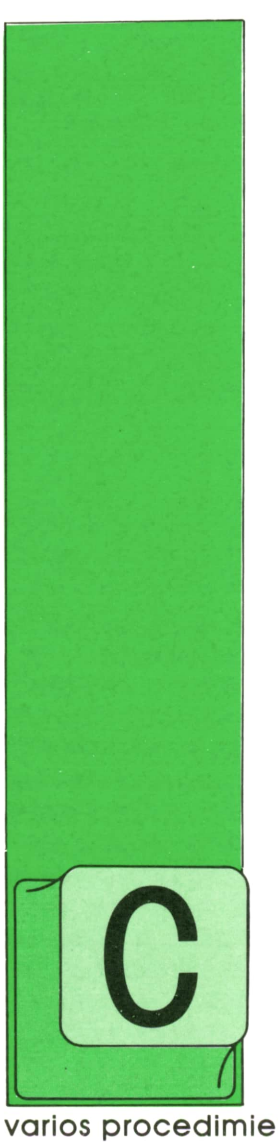
8. Obsérvese, por último, que dentro del programa principal se hace uso de la función o módulo, utilizando simplemente su nombre (MEDIA) seguido por una lista de valores entre paréntesis, tantos como los que hemos definido en el encabezamiento de la función, con tipos compatibles a los indicados allí.

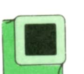
Veamos cuál es el resultado de la ejecución de nuestro programa principal:



Se observará que es exactamente igual que en el caso Basic. Sin embargo, supongamos que deseamos trasplantar la función MEDIA a otro programa principal distinto. No hay ningún problema. En efecto, todas las variables utilizadas por MEDIA son locales, por lo que el programa principal podría utilizar los nombres i, total, N y X para otras cosas, sin entrar en conflicto con el módulo. Además, aquí no tenemos el problema del conflicto en la numeración, que en BASIC se da con frecuencia, pues en Pascal no se utilizan casi los números de etiqueta (en el ejemplo anterior, ni una sola vez).

LOGO





Algunos comandos para procedimientos y variables

CUANDO aprendimos lo que son los procedimientos vimos algunos comandos que se pueden utilizar con ellos:

— BORRA o BO

Para borrar uno o varios procedimientos de la memoria de la tortuga.

— BOTODO

Para borrar todos los procedimientos definidos.

— IM

Para ver la definición de uno o varios procedimientos.

— IMTODO

Para ver la definición de todos los procedimientos.

— IMTS

Para ver los nombres de todos los procedimientos.

Al dar la definición de estos comandos no habíamos visto aún lo que son las variables. Como ahora ya las conocemos, hay que hacer una aclaración para los cuatro primeros.

El comando

BO

sirve para borrar de la memoria de la tortuga tanto procedimientos como variables y su valor correspondiente.

El comando

BOTODO

borra todos los procedimientos que se hayan definido y todas las variables utilizadas con HAZ junto con su valor asociado.

El comando

IM

permite ver la definición de procedimiento y el contenido (valor) de las variables.

El comando

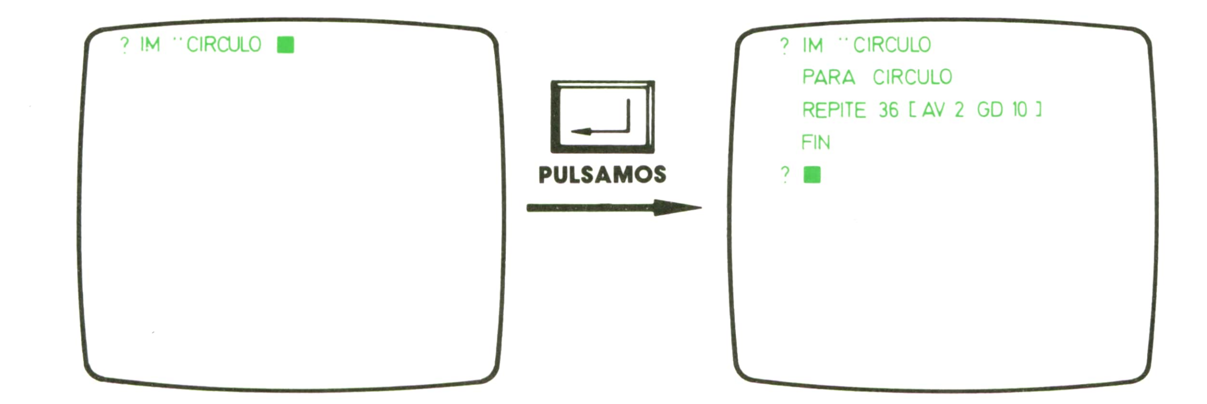
IMTODO

muestra la definición de todos los procedimientos y el contenido de todas las variables.

Por ejemplo, supongamos que hemos escrito lo siguiente:

```
? PARA CIRCULO
> REPITE 36 (AV 2 GD 10)
> FIN
? HAZ "DEPORTE "FUTBOL
? PARA ESTRELLA
> REPITE 10 (AV 35 RE 35 GD 36)
> OT
> FIN
? HAZ "EDAD 7
? HAZ "LIBRO "QUIJOTE
```

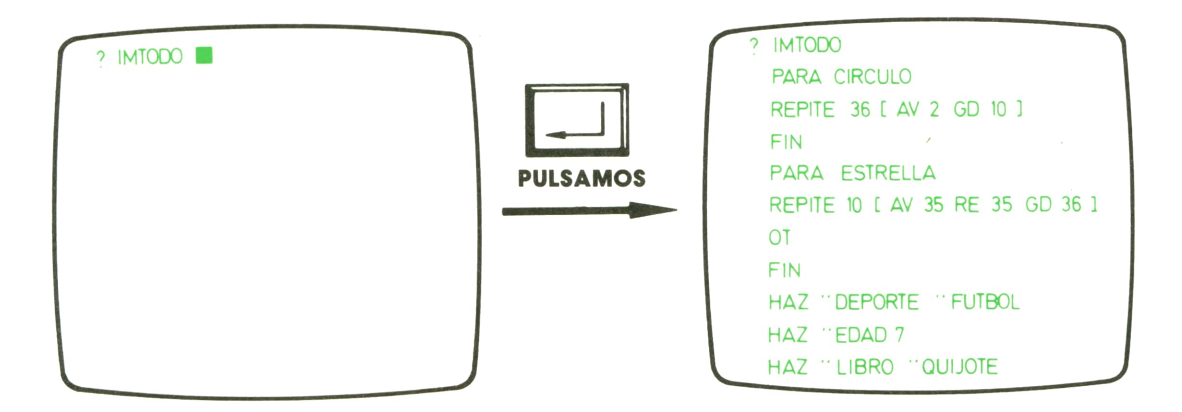
Vamos a ver lo que haría la tortuga al utilizar algunos de los comandos anteriores:



Como vemos, el nombre de la variable se ha de poner precedido por dos puntos (:).



Primero se nos muestran las definiciones de procedimientos y luego los valores de las variables.



Por último, un comando que sólo se puede usar con procedimientos es

COPIADEF "nombre1 "nombre2

(abreviatura de COPIA DEFinición), que sirve para copiar la definición del procedimiento **nombre1** en otro procedimiento llamado **nombre2.**

Puede ser muy útil cuando queremos definir dos procedimientos que son muy parecidos. En lugar de escribir la definición completa de los dos, podemos escribir la de uno y ellos y copiarla para el segundo. Luego, podemos utilizar el editor para modificar esta segunda definición.



Dos funciones nuevas

Ya sabemos que las funciones son unas órdenes que provocan que la tortuga nos dé una respuesta y que con esta respuesta tenemos que hacer algo. Por ejemplo, escribirla.

La función

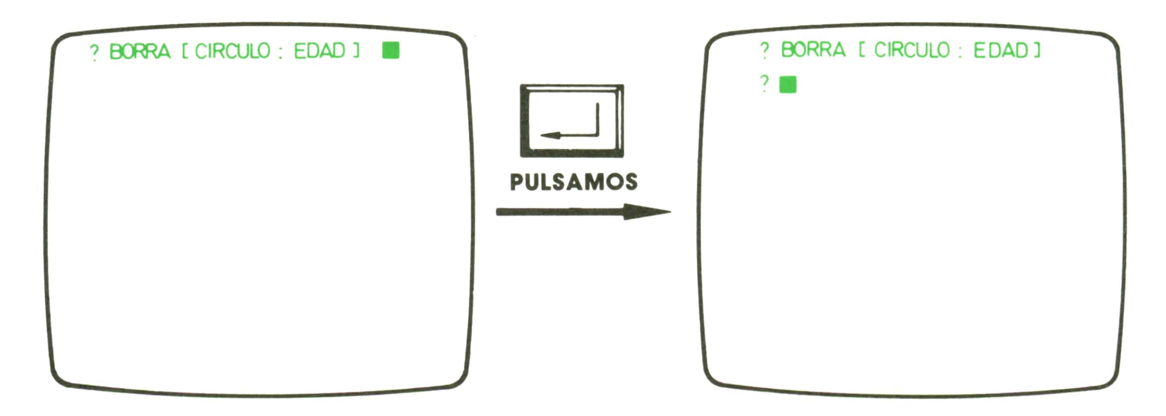
LPROCS

(abreviatura de Lista PROCedimientoS) devuelve una lista que contiene todos los nombres de procedimientos que hayamos definido, mientras que la función

LVARS

(abreviatura de Lista VARiableS) devuelve una lista de todos los nombres de variables que hemos utilizado.

Suponiendo que tenemos definidos los procedimientos ROMBO y AVION y que hemos asignado un valor a las variables COLOR, DIA y HORA, tendríamos que:



El resultado de estas funciones se puede utilizar, además, con otros comandos.

Si escribimos:

```
? GUARDA "PROCS LPROCS
? GUARDA "VARS LVARS
```

la tortuga copia en el fichero PROCS las definiciones de los dos procedimientos y en el fichero VARS los nombres y el valor de las tres variables.

O si ponemos:

```
? BO LPROCS
? BO LVARS
```

con el primer comando BO borramos de la memoria de la tortuga todos los procedimientos y con el segundo borramos todas las variables y sus valores correspondientes.



Cómo usar la impresora

Hasta ahora, cada vez que hemos pedido a la tortuga alguna información, ésta nos ha salido por la pantalla. Si lo que queremos es obtener lo mismo pero que nos salga por la impresora, primero tendremos que activarla, es decir, indicarle a la tortuga que en este momento queremos que utilice la impresora para comunicarse con nosotros. Para esto se usa el comando

IMPRESORA 1

A partir de este momento, todo lo que escribimos o lo que aparezca en la pantalla se imprimirá por la impresora.

Cuando ya no queramos que nos saque las cosas por la impresora pondremos

IMPRESORA 0

para indicarle a la tortuga que desactive la impresora.

Es decir, en general, el comando de control de la impresora es

IMPRESORA n

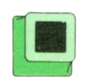
donde *n* es un *0* si queremos desactivarla o un *1* si queremos activarla.

Así, por ejemplo, si queremos sacar por impresora la definición de todos los procedimientos que tenemos y luego seguir trabajando normalmente, escribiremos:

```
? IMPRESORA 1
? IM LPROCS
```

Si ahora queremos imprimir solamente los nombres de los procedimientos y, además, todos los nombres de las variables y sus valores correspondientes, habrá que poner:

```
? IMPRESORA 1
? IMTS
? IM LVARS
? IMPRESORA 0
```

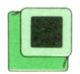


Números

Todos sabemos que una de las características que más llaman la atención de un ordenador es que puede realizar muchas y diferentes operaciones de una forma rápida y sin equivocarse.

Nuestra amiga la tortuga no va a ser menos y, como ya hemos visto en muchas ocasiones, es capaz de manejar números y de realizar distintas operaciones con ellos. Por ejemplo, utilizamos números para decirle a la tortuga el número de pasos que queremos que avance o el número de grados que ha de girar. También los usamos para meter un valor en el cajón correspondiente a una variable o para indicarle a la tortuga el color con que queremos que pinte su lápiz.

Vemos, pues, que, sin darnos cuenta, hemos estado utilizando números continuamente y que la tortuga no ha protestado en ningún momento.



Tipos de números

Principalmente, nuestra tortuga entiende tres tipos de números:

— *Números positivos*

Son los que más hemos utilizado hasta ahora. No llevan ningún carácter delante. Por ejemplo: 5, 123.

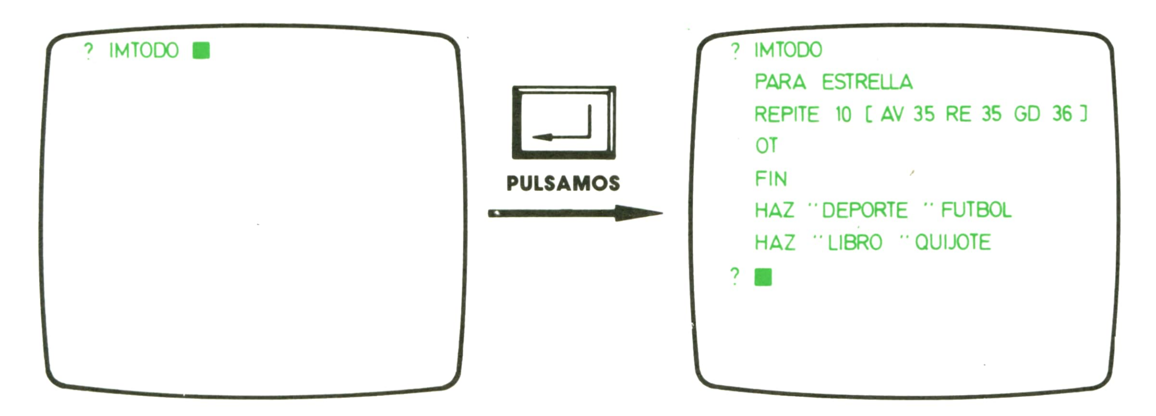
— *Números negativos*

Llevan delante un signo menos (−). Por ejemplo: −44, −12.

— *Números decimales*

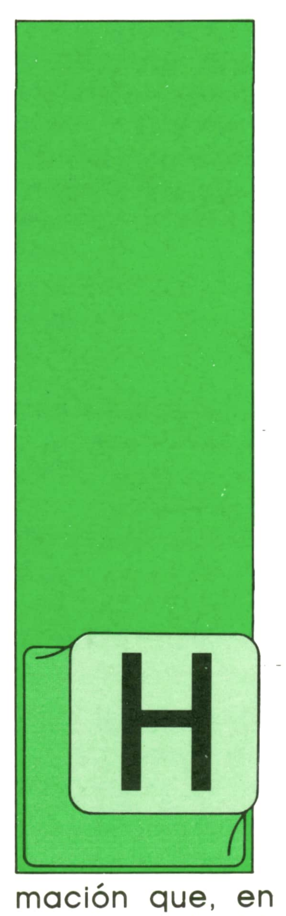
Son los que tienen parte decimal. Esta parte se separa de la parte entera por un punto (.). A su vez, pueden ser positivos y negativos. Por ejemplo: 0.7, 25.175, −3.94.

Como es lógico, si nosotros escribimos en pantalla un número sólo, la tortuga nos dirá que no sabe qué hacer con él. Es decir, tenemos que usar los números para algo. Puede ser para acompañar a diferentes comandos, como AV, RE, GD, GI, REPITE, PONCL, PONFONDO, ..., simplemente para escribirlos por medio del comando ES,

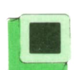


o para realizar operaciones con ellos.

PASCAL



HACE tiempo escribimos un programa para jugar al Master Mind que, tras generar una combinación aleatoria, nos permitía intentar descubrirla a través de la información que, en forma de número de muertos y heridos, nos devolvía como respuesta a los sucesivos intentos. Mencionamos entonces la posibilidad de escribir un programa por medio del cual fuera el ordenador el que intentase descubrir una combinación secreta ideada por nosotros y dejamos su realización para otro momento; pues bien, ese momento ha llegado.



El juego del Master Mind (2)

Cuando describimos la versión de Master Mind que íbamos a utilizar, pusimos un ejemplo de partida mostrando un típico proceso deductivo de este juego; resulta evidente que la cantidad de posibles situaciones distintas es enorme, por lo que un programa que simulase nuestra forma de razonar sería terriblemente complejo. El mismo problema, aunque a una escala mucho mayor, se presenta a la hora de escribir un programa para jugar al ajedrez.

Por fortuna, tanto en este último caso como en el que nos ocupa, podemos escribir programas relativamente sencillos y aprovechar la gran velocidad de los ordenadores para suplir la falta de «inteligencia» con la capacidad de analizar una gran cantidad de jugadas en muy poco tiempo; en otras palabras, es como resolver un problema de matemáticas por «la cuenta de la vieja», pero muy deprisa. Los resultados que se pueden obtener así es posible que los haya observado el lector con alguno de los juegos de ajedrez existentes actualmente en el mercado y, cuando ponga a prueba el programa que vamos a escribir, verá con el Master Mind cómo el ordenador puede parecer realmente inteligente.

El método, muy sencillo, consiste en lo siguiente: sólo se presenta una combinación al oponente del ordenador si es factible como solución de la partida, es decir, si, comparada con todos los anteriores intentos, da como resultado en cada caso exactamente el número de muertos y heridos que se le dieron al ordenador como respuesta. A medida que avanza la partida, el número de condiciones que es necesario cumplir para ser una combinación factible va aumentando y, por tanto, cada vez es menor el abanico de posibles combinaciones donde escoger, hasta llegar a un momento en que no hay más que una, que es precisamente la solución de la partida. El resultado, como observará el lector, es difícilmente diferenciable del obtenido con un proceso auténticamente deductivo y, en promedio, proporciona unos resultados próximos a los mejores obtenibles por un ser humano.

A la hora de buscar una combinación podemos seguir un método aleatorio, o bien un método sistemático; este último tiene la ventaja sobre el primero de que obtiene combinaciones factibles dentro de un plazo límite aunque las condiciones a cumplir sean muy restrictivas, teniendo el inconveniente, sin embargo, de que cuando una partida está en sus comienzos, al no haber prácticamente condiciones, las combinaciones que se presentan son casi siempre más o menos las mismas, perdiéndose, además, el factor suerte que siempre puede ser beneficioso al principio. Por otra parte, si, tras haberse explorado todas las combinaciones (de lo cual sólo se puede llegar a estar seguro con el método sistemático), no se ha obtenido la solución, el programa puede deducir que ha habido alguna respuesta incorrecta por parte de su oponente.

Lo más adecuado es, por tanto, empezar utilizando un método aleatorio de generación de combinaciones, que se irán contrastando con todos los intentos anteriores hasta encontrar alguna factible. Solamente cuando se haya probado infructuosamente un determinado número de combinaciones (en nuestro caso quinientos, aunque esto depende del gusto de cada uno) se pasará a la generacion sistemática; la forma natural de implementar ésta es por medio de cuatro (en realidad, tantos como elementos tenga cada combinación) bucles FOR anidados unos dentro de otros que, con vistas a camuflar la sistematicidad de la exploración, no estarán puestos por orden ni tendrán el mismo sentido de variación de sus variables. Por supuesto, tras encontrar una combinación factible por este método, la exploración se reanuda en el punto en que se encontró, no se vuelve a empezar por el principio.

Cada intento, junto con las respuestas asociadas, se guardará en una ficha perteneciente a una tabla de éstas que denominaremos Archivo. A la hora de analizar la factibilidad de una combinación se empezará a comparar por el final del Archivo, pues, normalmente, los últimos intentos habrán obtenido respuestas más restrictivas (es decir, con números más significativos de muertos y heridos) y así se empezará a comparar por el final de sin interés; es una cuestión de ahorro de tiempo exclusivamente.

En el programa se emplean dos instrucciones GOTO; como verá el lector, su empleo simplifica apreciablemente el programa.

Como ejercicio, podría modificar el lector el programa para que permita el juego en las dos direcciones; las funciones Muertos_1_A_2 y Heridos_1_A_2 serían aprovechables en ambos casos.

```
program  MasterMind;

label 10;

type
 Combi_t = array [1..4] of integer;
 Ficha_t = record
             Clave    : Combi_t;
             Muertos,
             Heridos : integer
           end;

 var
  Aleatorio : real;

  Prueba1,
  Prueba2,
  Prueba3,
  Prueba4     : integer;
  Prueba,
  Clave1,
  Clave2      : Combi_t;

  Archivo     : array [1..20] of Ficha_t;

  Ultima,
  Limite,
  FallosAzar: integer;
  OK          : boolean;
  Letra       : char;
```

```
(*===========================================================================*)
procedure ArrancaAzar;
(*---------------------------------------------*)
(* Pide el primer número de la serie aleatoria *)
(*---------------------------------------------*)
  var Bien: boolean;
 begin
  writeln ('Teclee un número entre 0 y 1, ambos exclusive.');
  repeat
   readln (Aleatorio);
   Bien:= (0.0 < Aleatorio) and (Aleatorio < 1.0);
   if not Bien then writeln ('No vale. Repita.')
  until Bien
 end;

(*===========================================================================*)
function Azar (Inf,Sup: integer): integer;
(*---------------------------------------------*)
(*   Proporciona un número entero aleatorio    *)
(*   entre Inf y Sup, ambos inclusive          *)
(*---------------------------------------------*)
  var A: real;
 begin
  Aleatorio:= frac (Aleatorio * 997);              (* ¡OJO! *)
  Azar:= Inf + trunc (Aleatorio * (Sup - Inf + 1))
 end;

(*===========================================================================*)
function MUERTOS_1_A_2: integer;
(*------------------------------------------------------------------*)
(* Compara Clave1 con Clave2 y devuelve el número de Muertos. *)
(*------------------------------------------------------------------*)
  var I,M: integer;
begin
 M:= 0;
 for I:= 1 to 4 do
   if Clave1 [I] = Clave2 [I] then
     begin
      Clave1 [I]:= -1;
      Clave2 [I]:= -2;
      M:= M + 1
     end;
 Muertos_1_a_2 := M
end;

(*===========================================================================*)
function HERIDOS_1_A_2: integer;
(*------------------------------------------------------------------*)
(* Compara Clave1 con Clave2 y devuelve el número de Heridos. *)
(*        DEBE utilizarse después de Muertos_1_a_2.                 *)
(*------------------------------------------------------------------*)
 var I,J,H: integer;
begin
 H:= 0;
 for I:= 1 to 4 do
   for J:= 1 to 4 do
     if Clave1 [I] = Clave2 [J] then
       begin
        Clave1 [I]:= -1;
        Clave2 [J]:= -2;
        H:= H + 1
       end;
 Heridos_1_a_2 := H
end;

(*===========================================================================*)
function ACEPTABLE: boolean;
  var Acept: boolean; Indice: integer;
(*------------------------------------------------------------------*)
(*  Compara Prueba con todas las combinaciones guardadas en   *)
(*  Archivo para ver si es una combinación factible.           *)
(*------------------------------------------------------------------*)
begin
 Acept  := true;     (* Suponemos, en principio, que Prueba es válida *)
Indice := Ultima;   (* Empieza a mirar por el final de Archivo       *)
(*----------------------------------------*)
(* Explora hasta el comienzo de archivo o *)
(* hasta encontrar alguna incongruencia:  *)
(*----------------------------------------*)
while (Indice > 0) and Acept do
  with Archivo [Indice] do
    begin
     (* Saca copias de las combinaciones a comparar: *)
```

```
      Clave1 := Clave;
      Clave2 := Prueba;

      (* Mira a ver si el resultado de su comparación está de *)
      (* acuerdo con la respuesta que se dio en su momento:    *)

      Acept  := (Muertos_1_a_2 = Muertos) and (Heridos_1_a_2 = Heridos);

      (* Pasa a otra combinación de Archivo: *)
      Indice  := Indice - 1
    end;
  (*-------------------------------------------*)
  (* Al llegarse aquí, si no se ha encontrado  *)
  (* ningún problema, Acept sigue siendo true: *)
  (*-------------------------------------------*)
  Aceptable := Acept

end;

(*=====================================================================*)
procedure MOSTRAR;
  var I,M,H: integer; Vale: boolean;
(*-------------------------------------------------------*)
(*  Enseña Prueba y pide el número de Muertos y Heridos,  *)
(*  guardando Prueba y las respuestas en Archivo.         *)
(*  Además, hace OK igual a TRUE si Prueba es la solución *)
(*-------------------------------------------------------*)
begin
 for I := 1 to 4 do write (Prueba [I]);

 (* Pide Muertos y heridos: *)
 repeat
   writeln;
   write ('      Muertos: '); read (M);
   write ('   Heridos: '); read (H);

   (* Comprueba coherencia de las respuestas: *)
   case M of
     0,1,2: Vale := (H >= 0) and (H <= 4-M);
     3,4  : Vale := (H = 0)
     else   Vale := false
    end;
   if not Vale then write ('      ¡Imposible!')

 until Vale;

 OK := (M = 4);
 if not OK then
   begin
    writeln ('      Pensando...');
    Ultima := Ultima + 1;   (* ya hay un intento más *)
    Archivo [Ultima].Clave    := Prueba;
    Archivo [Ultima].Muertos := M;
    Archivo [Ultima].Heridos := H
   end
end;

(*=====================================================================*
begin
 ClrScr;
 ArrancaAzar;
 write ('Números del 1 al...');
 readln (Limite);
 (*-------------------------------------------------- repetir partidas: *)
 repeat
  ClrScr;
  write ('Piense una combinación de cuatro cifras entre 1 y ',Limite);
  writeln (' y pulse Intro después.');
  readln;
  Ultima:= 0;
  FallosAzar:= 0;

  (*------------------------------ repetir búsqueda aleatoria:    *)
  repeat
    Prueba [1] :=  Azar (1, Limite);
    Prueba [2] :=  Azar (1, Limite);
    Prueba [3] :=  Azar (1, Limite);
    Prueba [4] :=  Azar (1, Limite);

    if Aceptable then
        begin
         Mostrar;
         if OK then GOTO 10;
         FallosAzar:= 0
```

```
        end
      else
       FallosAzar := FallosAzar + 1;

  until FallosAzar = 500;

  (*------------------------------ iniciar búsqueda sistemática: *)
  for Prueba4 := 1 to Limite do
    for Prueba2 := Limite downto 1 do
      for Prueba3 := 1 to Limite do
        for Prueba1 := Limite downto 1 do
          begin
           Prueba [1] := Prueba1;
           Prueba [2] := Prueba2;
           Prueba [3] := Prueba3;
           Prueba [4] := Prueba4;
           if Aceptable then
             begin
              Mostrar;
              if OK then GOTO 10
             end
          end;

  (* Si se llega aquí, es porque se han probado  *)
  (* todas las combinaciones sin éxito:          *)

  writeln; writeln ('REVISE SUS RESPUESTAS');

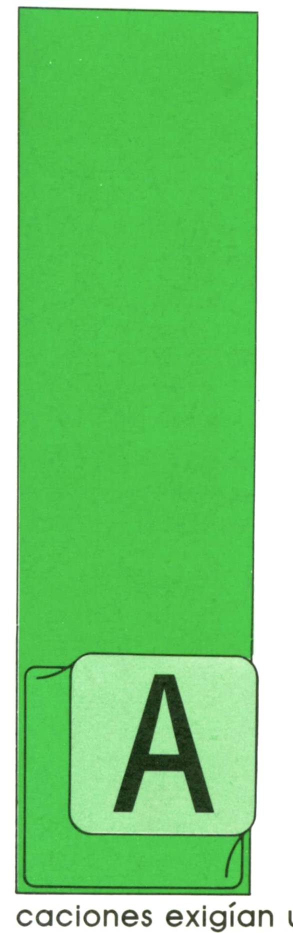
10: writeln; writeln; write ('Otra partida (S/N) ? '); readln (Letra)

 until (Letra = 'N') or (Letra = 'n')

end.
```

OTROS LENGUAJES

EL LENGUAJE DE PROGRAMACION ADA



ADA es un lenguaje que fue diseñado de encargo, basándose en las especificaciones Steelman del Departamento de Defensa de los Estados Unidos. Estas especificaciones exigían un lenguaje ante todo potente, diseñado para su implementación en grandes sistemas de cálculo, y que fuese además utilizable en problemas muy diferentes, es decir, un lenguaje de programación de uso general.

Debido a esto, ADA ofrece una serie de facilidades, como las de lenguajes clásicos de programación como Pascal, así como otras frecuentemente utilizadas en lenguajes especializados. Por ello es un lenguaje de tipo algorítmico, que además de poseer las estructuras típicas de control, como la definición de tipos, subprogramas, etc., dispone de las características de los lenguajes de vanguardia, como modularidad; donde tipos, datos y subprogramas pueden ser empaquetados.

Este lenguaje trata la modularidad también en el sentido físico, facilitando la compilación por partes, incluso en diferentes máquinas.

Además de estas características, el lenguaje permite la programación para atender a tareas en tiempo real, facilitando de base, ejecución en paralelo y el manejo de excepciones (interrupciones y señales asíncronas de todo tipo).

Las operaciones de entrada/salida están definidas a ambos niveles: aplicación y máquina.

Cuando el lenguaje ADA fue diseñado, se hizo basándose en tres conceptos principalmente:

— Reconocimiento de la importancia del mantenimiento en el rendimiento del software.

— Facilitar la tarea de programación basándose en las características tipo de los programadores humanos (reconocimiento de errores, etc.).

— Conseguir, ante todo, un lenguaje eficiente.

Debido a esto, el resultado es un lenguaje cuyas posibilidades y profundidad sintáctica es enorme, para hacer posible toda esta potencia; al mismo tiempo, sus reglas son muy estrictas para impedir al máximo los frecuentes errores de tipo, en la asignación de datos entre bloques diferentes, muy habituales en los lenguajes de programación menos exigentes.

Como consecuencia, al ser un lenguaje tan extenso, la mayoría de los compiladores existentes sólo tratan un pequeño subconjunto de instrucciones del lenguaje. Sólo con ordenadores de gran potencia es posible implementar este lenguaje, que es, sin duda, de los más potentes en la actualidad.

Veamos brevemente cuál es la estructura de un programa ADA típico.

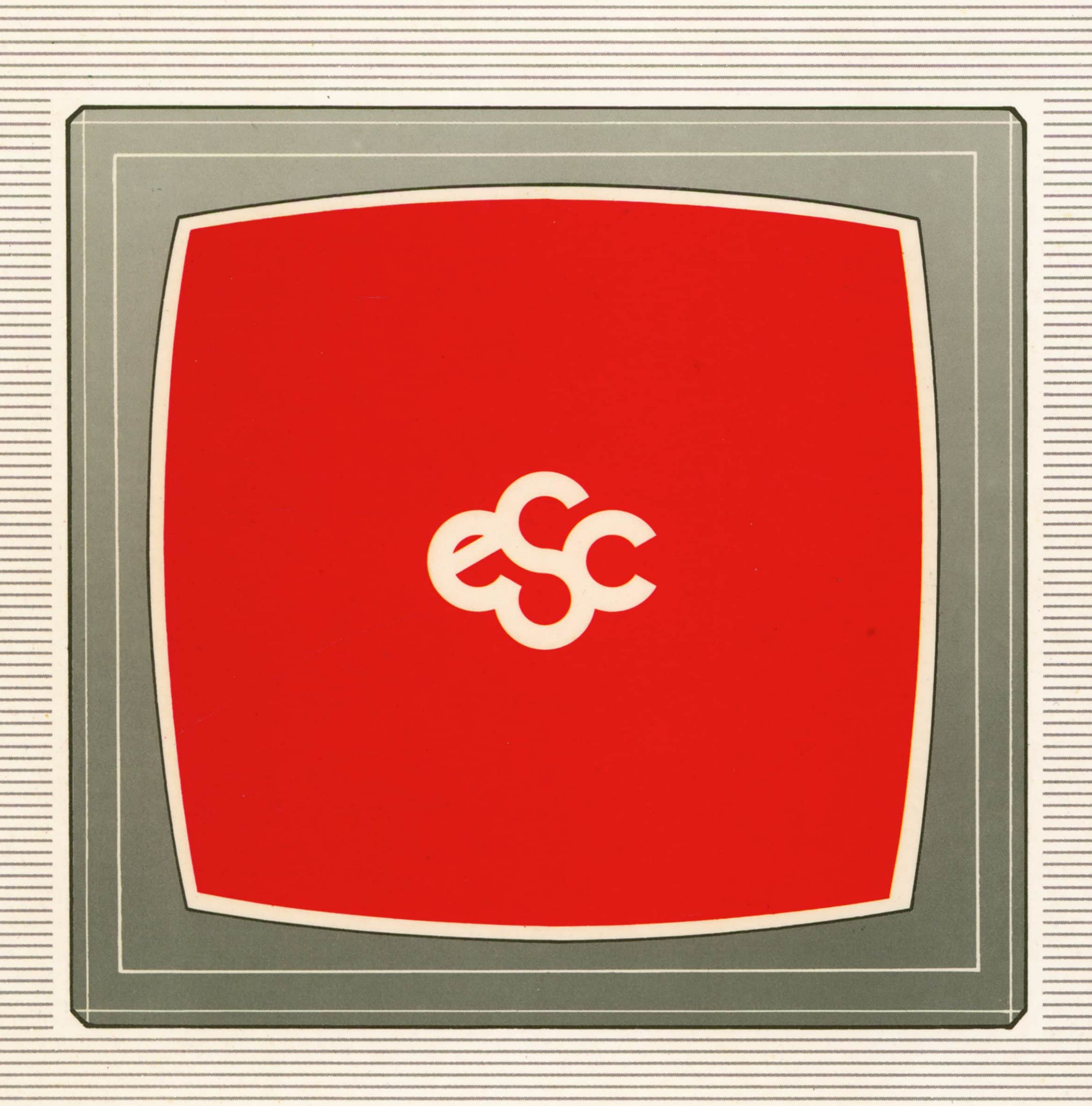
El siguiente programa ADA es una unidad compilable, un procedimiento que

imprime las raíces reales de una ecuación de segundo grado:

```
with TEXT_10,REAL_OPERATIONS; use REAL_OPERATIONS;
procedure QUADRATIC_OPERATION is
  A,B,C,D:REAL;
  use REAL_IO, -- define GET y PUT para REAL
      TEXT_IO, -- define PUT para STRING y NEW_LINE
      REAL_FUNCTIONS, -- define SQRT


  begin
    GET(A);
    GET(B);
    GET(C);
    D := B ** 2 - 4.0 * A * C;
    if D < 0.0 then
       PUT (" Raices imaginarias ");
    else
       PUT (" Raices reales: X1= ");
       PUT ((- B - SQRT ( D ) ) / ( 2.0 * A ) );
       PUT (" X2= ");
       PUT ((- B + SQRT ( D ) ) / ( 2.0 * A ) );
    endif;
    NEW_LINE;
end CUADRATIC_EQUATION;
```

Como puede observarse, la sintaxis tiene un gran parecido con la del lenguaje MODULA-2, que es una evolución modular del Pascal.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼